C++17新特性带来的改善和影响

吴钊
IBM

Legal Disclaimer

- This work represents the view of the author and does not necessarily represent the view of IBM.

- IBM, PowerPC and the IBM logo are trademarks or registered trademarks of IBM or its subsidiaries in the United States and other countries.

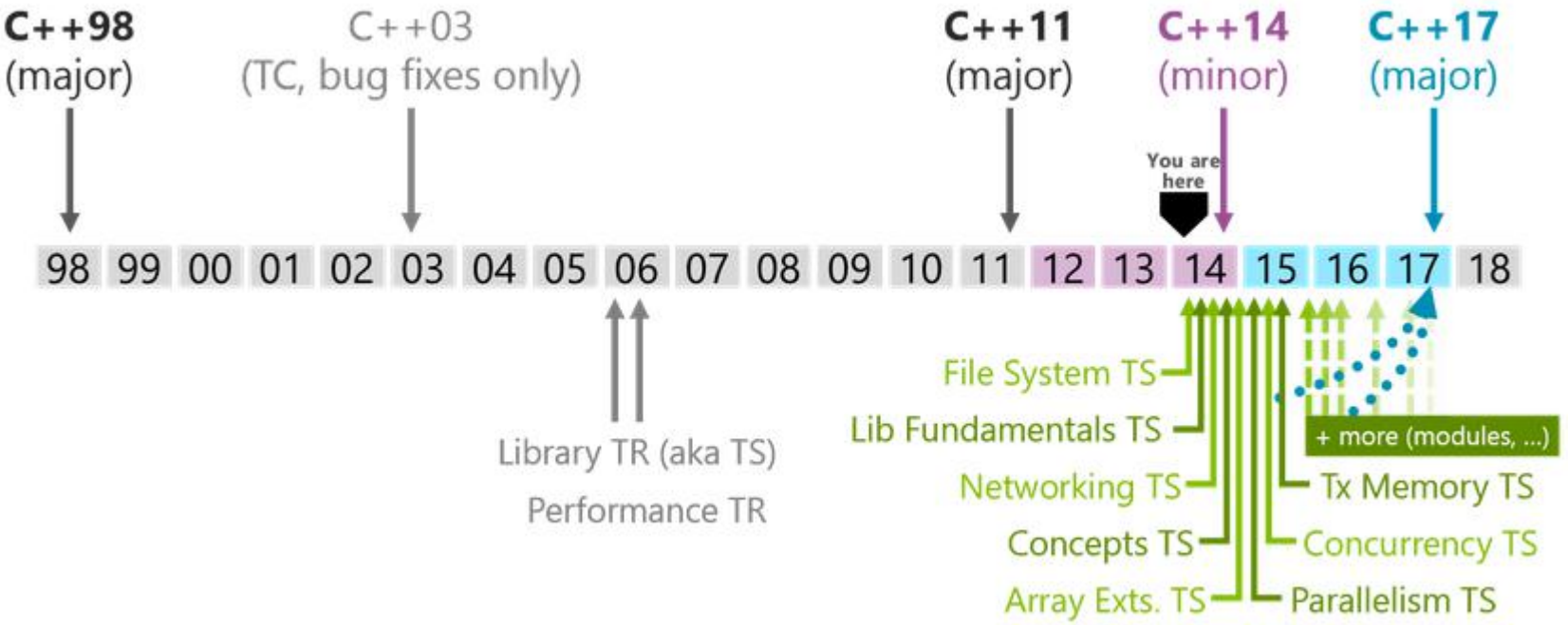
- Other company, product, and service names may be trademarks or service marks of others.

主要内容

- Concepts
- Module System
- C++17状态

内容是如此的“充实”...

C++ Timeline


C++98
(major)
C++03
(TC, bug fixes only)
C++11
(major)
C++14
(minor)
C++17
(major)
You are here
98 99 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18
Library TR (aka TS)
Performance TR
File System TS
Lib Fundamentals TS
Networking TS
Concepts TS
Array Exts. TS
Tx Memory TS
Concurrency TS
Parallelism TS
+ more (modules, ...)

Case #1

```
template<typename R, typename T>
  bool exist(const R& range, const T& value) {
    for (const auto& x : range)
      if (x == value)
        return true;
    return false;
  }

vector<string> vec {};
exist(vec, 47); // BUG!
```

Case #1

- GCC 5.2:

```
main.cpp: In instantiation of 'bool exist(const R&, const T&) [with R =
    std::vector<std::__cxx11::basic_string<char> >; T = int]':
main.cpp:16:18:   required from here
main.cpp:8:13: error: no match for 'operator==' (operand types are
    'const std::__cxx11::basic_string<char>' and 'const int')
      if (x == value)
            ^
In file included from /usr/local/include/c++/5.2.0/bits/stl_algobase.h:64:0,
                 from /usr/local/include/c++/5.2.0/bits/char_traits.h:39,
                 from /usr/local/include/c++/5.2.0/string:40,
                 from main.cpp:1:
```

报错信息接下一页...

```
/usr/local/include/c++/5.2.0/bits/stl_pair.h:214:5: note: candidate: template<class _T1, class _T2> constexpr bool std::operator==(const std::pair<_T1, _T2>&, const std::pair<_T1, _T2>&)
    operator==(const pair<_T1, _T2>& __x, const pair<_T1, _T2>& __y)
    ^
/usr/local/include/c++/5.2.0/bits/stl_pair.h:214:5: note:   template argument deduction/substitution failed:
main.cpp:8:13: note:   'const std::__cxx11::basic_string<char>' is not derived from 'const std::pair<_T1, _T2>'
      if (x == value)
            ^
In file included from /usr/local/include/c++/5.2.0/bits/stl_algobase.h:67:0,
                 from /usr/local/include/c++/5.2.0/bits/char_traits.h:39,
                 from /usr/local/include/c++/5.2.0/string:40,
                 from main.cpp:1:
/usr/local/include/c++/5.2.0/bits/stl_iterator.h:292:5: note: candidate: template<class _Iterator> bool std::operator==(const std::reverse_iterator<_Iterator>&, const std::reverse_iterator<_Iterator>&)
    operator==(const reverse_iterator<_Iterator>& __x,
    ^
/usr/local/include/c++/5.2.0/bits/stl_iterator.h:292:5: note:   template argument deduction/substitution failed:
main.cpp:8:13: note:   'const std::__cxx11::basic_string<char>' is not derived from 'const std::reverse_iterator<_Iterator>'
      if (x == value)
            ^
In file included from /usr/local/include/c++/5.2.0/bits/stl_algobase.h:67:0,
                 from /usr/local/include/c++/5.2.0/bits/char_traits.h:39,
                 from /usr/local/include/c++/5.2.0/string:40,
                 from main.cpp:1:
/usr/local/include/c++/5.2.0/bits/stl_iterator.h:342:5: note: candidate: template<class _IteratorL, class _IteratorR> bool std::operator==(const std::reverse_iterator<_Iterator>&, const std::reverse_iterator<_IteratorR>&)
    operator==(const reverse_iterator<_IteratorL>& __x,
    ^
/usr/local/include/c++/5.2.0/bits/stl_iterator.h:342:5: note:   template argument deduction/substitution failed:
main.cpp:8:13: note:   'const std::__cxx11::basic_string<char>' is not derived from 'const std::reverse_iterator<_Iterator>'
      if (x == value)
            ^
In file included from /usr/local/include/c++/5.2.0/bits/stl_algobase.h:67:0,
                 from /usr/local/include/c++/5.2.0/bits/char_traits.h:39,
                 from /usr/local/include/c++/5.2.0/string:40,
                 from main.cpp:1:
/usr/local/include/c++/5.2.0/bits/stl_iterator.h:1065:5: note: candidate: template<class _IteratorL, class _IteratorR> bool std::operator==(const std::move_iterator<_Iterator>&, const std::move_iterator<_IteratorR>&)
    operator==(const move_iterator<_IteratorL>& __x,
    ^
/usr/local/include/c++/5.2.0/bits/stl_iterator.h:1065:5: note:   template argument deduction/substitution failed:
main.cpp:8:13: note:   'const std::__cxx11::basic_string<char>' is not derived from 'const std::move_iterator<_Iterator>'
      if (x == value)
            ^
In file included from /usr/local/include/c++/5.2.0/bits/stl_algobase.h:67:0,
                 from /usr/local/include/c++/5.2.0/bits/char_traits.h:39,
                 from /usr/local/include/c++/5.2.0/string:40,
                 from main.cpp:1:
/usr/local/include/c++/5.2.0/bits/stl_iterator.h:1071:5: note: candidate: template<class _Iterator> bool std::operator==(const std::move_iterator<_Iterator>&, const std::move_iterator<_Iterator>&)
    operator==(const move_iterator<_Iterator>& __x,
    ^
/usr/local/include/c++/5.2.0/bits/stl_iterator.h:1071:5: note:   template argument deduction/substitution failed:
main.cpp:8:13: note:   'const std::__cxx11::basic_string<char>' is not derived from 'const std::move_iterator<_Iterator>'
      if (x == value)
            ^
In file included from /usr/local/include/c++/5.2.0/bits/char_traits.h:40:0,
                 from /usr/local/include/c++/5.2.0/string:40,
                 from main.cpp:1:
/usr/local/include/c++/5.2.0/bits/postypes.h:216:5: note: candidate: template<class _StateT> bool std::operator==(const std::fpos<_StateT>&, const std::fpos<_StateT>&)
    operator==(const fpos<_StateT>& __lhs, const fpos<_StateT>& __rhs)
    ^
/usr/local/include/c++/5.2.0/bits/postypes.h:216:5: note:   template argument deduction/substitution failed:
main.cpp:8:13: note:   'const std::__cxx11::basic_string<char>' is not derived from 'const std::fpos<_StateT>'
      if (x == value)
            ^
In file included from /usr/local/include/c++/5.2.0/string:41:0,
                 from main.cpp:1:
/usr/local/include/c++/5.2.0/bits/allocator.h:128:5: note: candidate: template<class _T1, class _T2> bool std::operator==(const std::allocator<_CharT>&, const std::allocator<_T2>&)
    operator==(const allocator<_T1>&, const allocator<_T2>&)
    ^
/usr/local/include/c++/5.2.0/bits/allocator.h:128:5: note:   template argument deduction/substitution failed:
main.cpp:8:13: note:   'const std::__cxx11::basic_string<char>' is not derived from 'const std::allocator<_CharT>'
      if (x == value)
            ^
In file included from /usr/local/include/c++/5.2.0/string:41:0,
                 from main.cpp:1:
```

Case #2

```
template <typename T>
void f(T t) {
    // do some things...
    t.foo();
   // do some things...
}
```

一切都很好...

Case #2

一段时间后，因为需求，f需要完成一个功能，于是T加上了成员函数bar

```
template <typename T>
void f(T t) {
    // do some things...
    t.foo();
    // do some things...
    t.bar();
}
```

那么，与代码对应的文档描述呢？

Case #3

```
template <typename Container>
void f(const Container & C) {
    C. // 为什么我的IDE不提示容器Container的方法？如size
}
```

Case #4

```
vector<int> v { ... };
multiset<int> s { ... };
auto vi = find(v, 7); // calls sequence overload
auto si = find(s, 7); // calls associative container overload
```

如何区分是顺序容器还是关联容器，从而调用不同的容
find方法？SFINAE？

Why???

C++的模板参数没有语意

Generics in C#, Java, and C++

- C#
  - 如同class，但是含有类型参数
  - 强类型
  - 不需要向下转换
  - 与之前的CLR不兼容
  - 类型信息传递给CLR
  - Runtime实例化

```
interface IHasName{ string Name(); };
string addNames<T>( T first, T second ) where T : IHasName
    { return first.Name() + second.Name();}
```

Generics in C#, Java, and C++

- Java
  - Project Pizza
  - 类型擦除
  - Java编译器帮你插入转换的代码
  - 与1.5之前的JVM兼容
  - 类型信息传进JVM

```
interface IHasName{ string Name(); };
String addNames<T extends IHasName>( T first, T
    second ) { return first.Name() + second.Name();}
```

Generics in C#, Java, and C++

- C++
  - 如同宏，却看起来像一个class
  - 生成原生的x86, PowerPC机器码
  - 编译期实例化
  - 类型参数只是占位，却不检测

  是否可以如同C# / Java 一样进行类型检测？？？

C++17 Answer:

Concepts!

Concept的故事...

Eric Niebler
@ericniebler

Following

The Concepts TS was voted out today! Concepts are (almost) an ISO standard. Congrats, A. Sutton. This will change everything. @isocpp #cpp

RETWEETS 97 LIKES 44

4:58 AM - 21 Jul 2015

Concepts

- 如何让模版参数真正的具有语意？那就是程序之道：抽象
- Concepts的本质也在于了抽象二字，为模板类型“薄薄”的抽象出一个“概念”，定义约束规则，从而使得模板类型具有语意。
- C++17 New Keywords: concept & requires

concept and requires

- concept是一种对代码本身更深，更严谨的思考方式
- 那么让我们思考以下的一个问题（解决Case #1问题的第一步）
- 如何定义一个序列（Sequence）本身的元素是否可以进行相等性的比较？（int, string, YourClass?）

concept and requires

- 若序列中的元素可以进行相等性比较，那么该序列的参数一定可以使用 == 与 != 操作符进行比较，然后返回一个bool结果，否则无法返回一个bool结果，这就是约束规则与条件（Constraints）

- 那么，如果设两个参数分别为a与b，则可以类似表示为

  {a == b} -> bool;

  {a != b} -> bool;

concept and requires

- 而约束规则(Constraint)正是concept的根基，concept本质则正是为定义一系列的约束条件的模板函数

```
template <typename T>
concept bool  Equality_comparable() { // 约束条件}
```

concept and requires

- 当为函数加上concept关键字后，则意味着：
  - 函数是constexpr
  - 函数的返回值必须是bool
  - 函数不带有任何的参数
  - 函数的实现体为一系列的约束条件

- 而约束条件则是我们之前的 {a == b} -> bool 等形式，那么如何表示呢？

concept and requires

```
template <typename T>
concept bool  Equality_comparable() {
     return requires (T a, T b) {
          {a == b} -> bool;
          {a != b} -> bool;
     };
}
```

concept and requires

- requires 表达式 是一种
  - 语意约束的组合
  - 编译时完成
  - 嵌套的约束条件语句为一个合法的表达式或者相关联的类型
  - 约束条件语句可以带上一个返回类型，如我们这里的->bool

- 如约束条件语句{a == b} -> bool;意味着
  - a == b对所有的类型T元素都合法
  - 返回值必须为bool
  - 若实例化时无法编译，或者返回值为false，则结果返回false

concept and requires

- 那么我们针对Case #1的问题，即跨类型比较的问题（string与int, double与int），如何更进一步定义我们的concept呢？

```
template<typename R, typename T>
  bool exist(const R& range, const T& value) {
    for (const auto& x : range)
      if (x == value)
        return true;
    return false;
  }

vector<string> vec {};
exist(vec, 47); // BUG!
```

concept and requires

- 若两个不同的类型需要比较，则必须满足以下三点:
  - 我们需要比较的两个类型自身是可以用作 == 与 != 比较的
  - 这两个不同的类型需要有共同类型
    - double与int的共同类型是double
    - string与int没有共同类型
    - 可以使用C++11的std::common_type解决
  - 转换的共同类型需要支持 == 与 != 操作符，并且使用 == 与 != 操作符时，可以进行操作数的交换

concept and requires

```
template<typename T, typename U>
concept bool Equality_comparable() {
  return Equality_comparable<T>()  // previous single arg version
      && Equality_comparable<U>()
      && Common<T, U>()  // what is this?
      && requires (T t, U u) {
           { t == u } -> bool;
           { u == t } -> bool;
           { t != u } -> bool;
           { u != t } -> bool;
      };
}
```

concept and requires

```
// The 'Common_type' 代表一系列可变模板参数类型的共同类型
template<typename... Ts>
using Common_type = typename std::common_type<Ts...>::type;


// 若commpn_type存在，'Common' concept则被满足
template<typename... Ts>
concept bool  Common()  {
  return requires { typename Common_type<Ts...>; };
}
```

使用concept

- 使用concept有两种方式
  - 作为模板的类型参数说明
    - template<typename R, Equality_comparable<Value_Type<R>> T>
      bool exist(const R& range, const T& value)
    - Value_Type也是concept，含义为取出对象的值类型，如vector<int>，则是int（实现放在了 backup中，若感兴趣可以查看）
  - 放在模板声明处
    - template<typename R, typename T>
      requires Equality_comparable<T, Value_type<R>>()
      bool exist(const R& range, const T& value)
- 这两种方式完全等价

使用concept的效果

```
blue@blue-ubuntu:~/Documents/third-party$ gcc-cpp/bin/c++ -std=c++1z a.cpp
a.cpp: In function ‘int main()’:
a.cpp:48:17: error: cannot call function ‘bool exist(const R&, const T&) [with R
    exist(vec, 47);
                ^

a.cpp:39:8: note:   constraints not satisfied
   bool exist(const R& range, const T& value) {
        ^

a.cpp:39:8: note:   concept ‘Equality_comparable<int, origin::Value_type<std::ve
d::allocator<char> >, std::allocator<std::__cxx11::basic_string<char, std::char_
d
blue@blue-ubuntu:~/Documents/third-party$
```

- 更好更清晰的提示，错误清晰的指出约束条件不满足，并且只有短短的3行，而之前是158行

使用concept解决其它问题

- Case #2 （文档与代码分离）

  现在有了concept，代码即文档，你修改任意的T的成员，你都需要体现在concept的代码中，符合约束或者修改约束，而不是以前没有语意的T

- Case #3 （IDE感知问题）

  现在的concept带有着语意，如你的concept是容器，那么IDE知道你这是容器，那么就可以找出容器的方法出来

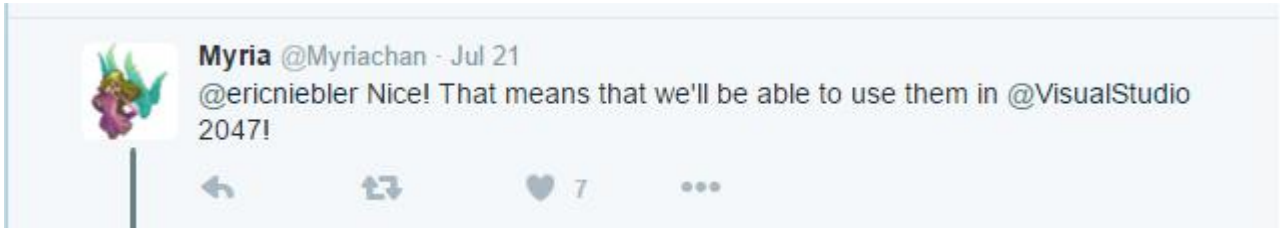
- Case #4 （重载问题）

  concept支持重载，如你所见Equality_comparable一般

When?

那么什么时候可以体验到呢？

2047?


Myria @Myriachan · Jul 21
@ericniebler Nice! That means that we'll be able to use them in @VisualStudio 2047!
7

Now!

- gcc-svn
  - svn://gcc.gnu.org/svn/gcc/trunk
- concepts library
  - https://github.com/asutton/origin

- Try and Enjoy it! :-)

Module System

- 无论是标准委员会，还是我们应用开发者，每一个人都想要的特性
- 它使C++从语言层面上支持组件化
- 它可以改善现有的C++程序编译速度
  - 组件化使得可以更好的利用分布式编译，云编译等
- 与宏（Macros）隔离开
- 它可以对代码分析，IDE等工具等更友好

Module所想要解决的问题

- C++最初名叫C with Classes (1979）
  - 保持对C代码级别的兼容性
- C++同时也继承了C的编译与链接模型

C/C++编译与链接模型

- 使用#include头文件包含模型（简单的拷贝复制，不基于语意）
  - 对C足够，然而C++模板与类的存在，使得代码量变得巨大
  - 试想使用iostream输出Hello World，编译器将会编译的代码量
- 程序由一系列的独立编译单元 (Tranlation Units, TU) 构成
  - 问题：每个编译单元都是独立的，不知道其它编译单元的信息
- 编译单元保持独立性，若需要通信，则需要使用外部名称 (External Names)
  - 问题：编译单元没有显示的指明依赖
- 链接器通过外部名称来找到相关符合的定义
  - 类型安全问题
  - 可能会违背ODR(One Definition Rule)

Module System

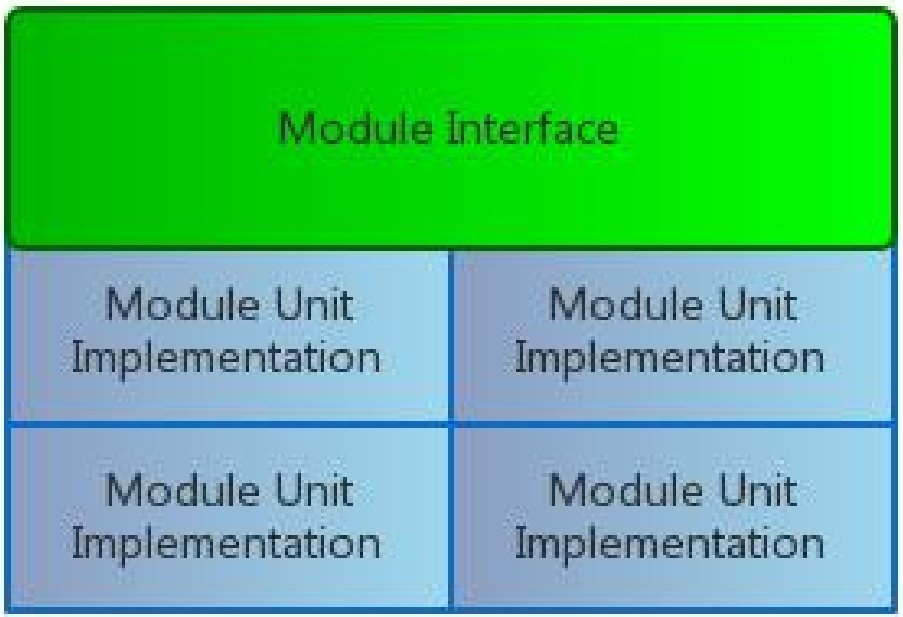
- Module 是一系列相关的编译单元（TU）的集合，该集合会提供一系列访问入口点。

- Module Interface
  - 一个模块可提供的声明

- Module Unit
  - 一个模块可提供的编译单元基础元素

- Module Name
  - 模块名，用于符号引用

Module System



MySome.module

如何表示Module

```
module M; // Module声明

// 想要暴露给使用Module的人
export {
    int f() { return 47 ;}
    // 你想要暴露的函数、class等
};

// 不属于Module M，只属于本地编译单元
int g() { return 48;}
```

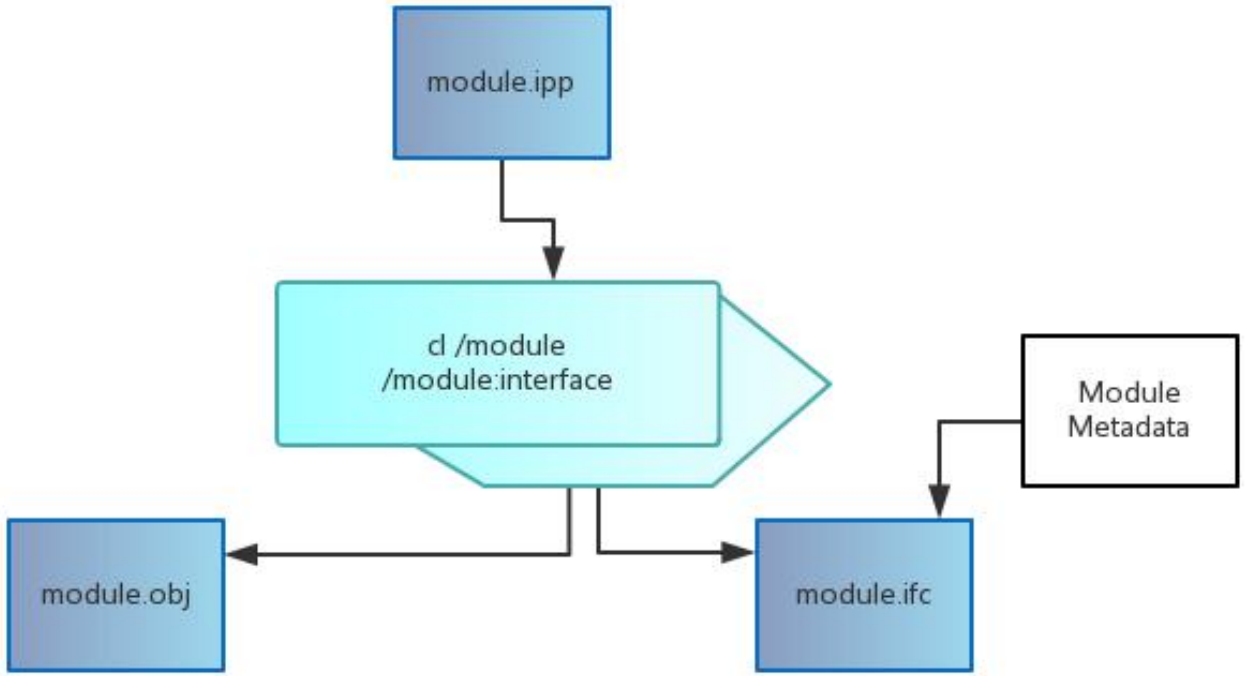
如何使用Module

```
import std.vector ; // #include <vector>
import std.iostream ; // #include <iostream>
import M;  // 导入我们的模块M
using namespace std;

int main() {
   vector<int> vec;
   f(); // OK, 模块可暴露提供的函数
   g(); // Error，该函数不可见！注意class私有成员则是不可访问却可见
}
```

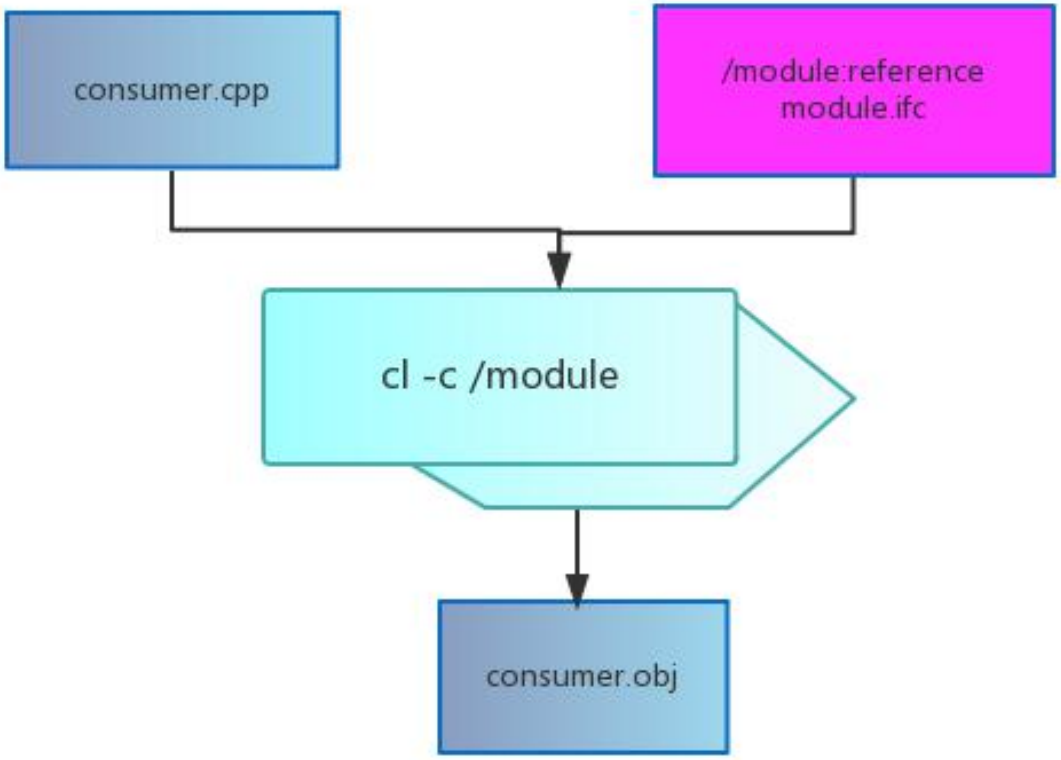
Module的可能实现方案（VC++）

- 编译模块接口



Module的可能实现方案（VC++）

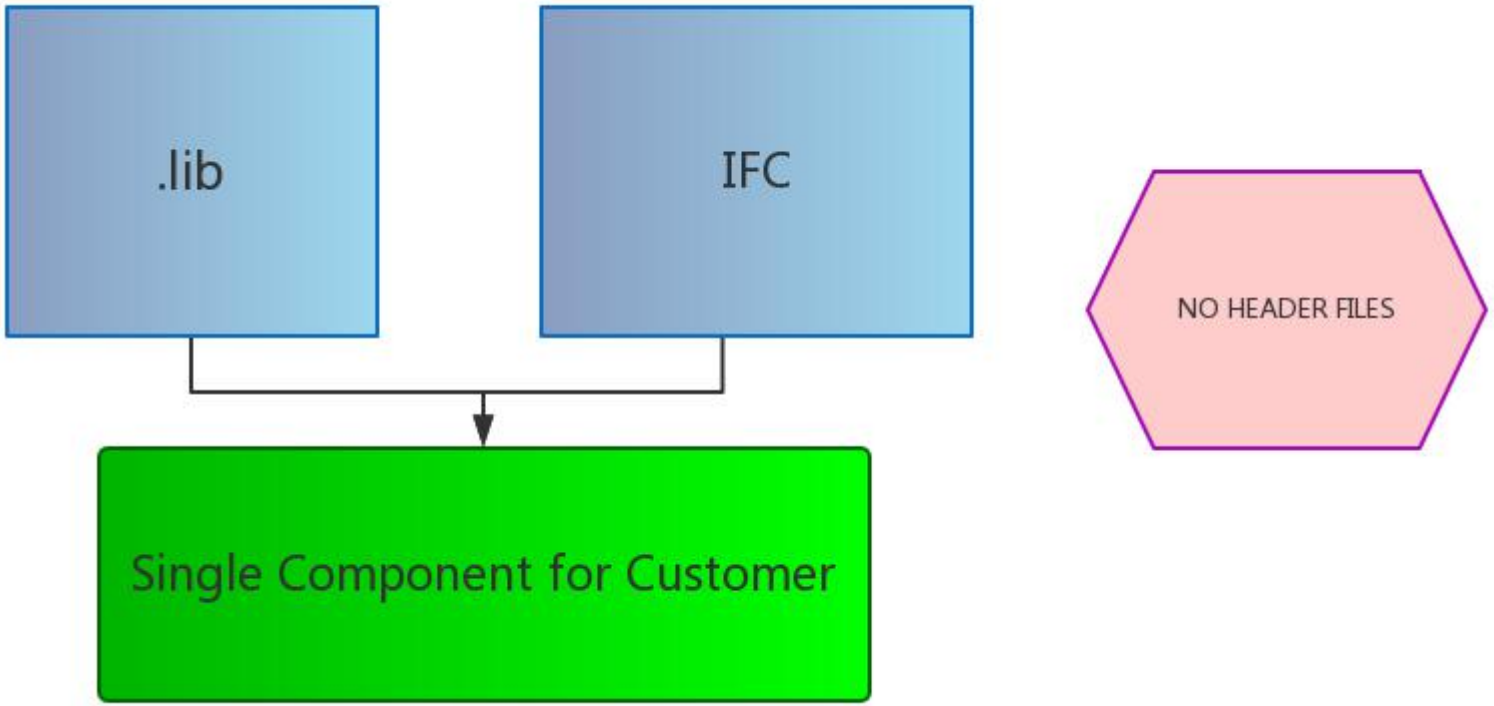
- 使用模块接口



Module的可能实现方案（VC++）

- /module
  - 开启module支持，如module等新关键字的支持
- /module: interface
  - 编译module interface，产生ifc文件
- /module: reference filename
  - 查找相关的module interface ifc文件
- IFC文件
  - 用来表示module interface的文件格式
  - 可被用来嵌套到静态/动态库
  - 格式开源

使用Module后的发布



Module vs PCH

- PCH（预编译，Pre-compiled Headers）
  - 是一个巨大的编译器Hack
  - #include pch，不会做任何预处理，编译等流程，而是使用已经导入到编译器的符号
  - 若你修改pch的一个文件，你必须重新编译整个巨大的PCH，这是PCH的受限处

- Module是PCH的“完美”形式
  - 不是编译器Hack, 语言提供支持，并且具有自己的表示形式，如IFC
  - IFC包含Module里面所需的编译单元符号在Object File等的相关位置等信息，所以你不再需要#include，只需要import，即可按需导入你所需的东西
  - 若你修改某个module的东西，只需要编译一个module即可，不需要再次编译其它module

When?

什么时候可以使用？

Now!

- Visual Studio 2015 Update 1

- 命令行编译Modules（我的知乎专栏）
  http://zhuanlan.zhihu.com/frozengene/20389770

C++17目前的状态(1/4)(截至10月)

Filesystems TS	Standard filesystem interface	Published!
Library Fundamentals TS I	optional, any, string_view and more	Published!
Library Fundamentals TS II	source code information capture and various utilities	Voted out for balloting by national standards bodies
Concepts (“Lite”) TS	Constrained templates	Publication imminent
Parallelism TS I	Parallel versions of STL algorithms	Published!
Parallelism TS II	TBD. Exploring task blocks, progress guarantees, SIMD.	Under active development

C++17目前的状态(2/4)

Transactional Memory TS	Transaction support	Published!
Concurrency TS I	improvements to future, latches and barriers, atomic smart pointers	Voted out for publication!
Concurrency TS II	TBD. Exploring executors, synchronic types, atomic views, concurrent data structures	Under active development
Networking TS	Sockets library based on Boost.ASIO	Design review completed; wording review of the spec in progress

C++17目前的状态(3/4)

Ranges TS	Range-based algorithms and views	Design review completed; wording review of the spec in progress
Numerics TS	Various numerical facilities	Beginning to take shape
Array Extensions TS	Stack arrays whose size is not known at compile time	Direction given at last meeting; waiting for proposals
Reflection	Code introspection and (later) reification mechanisms	Still in the design stage, no ETA
Graphics	2D drawing API	Waiting on proposal author to produce updated standard wording

C++17目前的状态(4/4)

Modules	A component system to supersede the textual header file inclusion model	Microsoft and Clang continuing to iterate on their implementations and converge on a design. The feature will target a TS, not C++17.
Coroutines	Resumable functions	At least two competing designs. One of them may make C++17.
Contracts	Preconditions, postconditions, etc.	In early design stage

Say Thanks!

I do this work because I love it and because I love C++ and want it to be as excellent as I know it can be. If you like my work and are looking for a way to say thank you, you can leave a supportive comment on my blog. Or you could leave me some kudos on my Open Hub range-v3 contribution page. Just click the **Give Kudos** button here.

Backup

Value_type Implementation: 也可参考github origin concepts library.

```
namespace core_impl
{
// 不同的类型
template<typename T>
  struct get_value_type;

template<typename T>
  struct get_value_type<T*> { using type = T; };

template<typename T>
  struct get_value_type<const T*> { using type = T; };

template<typename T>
  struct get_value_type<T[]> { using type = T; };
```

```
template<typename T>
  requires requires () { typename T::value_type; }
    struct get_value_type<T> { using type = typename T::value_type; };

// Make iostreams have a value type.
template<typename T>
  requires Derived<T, std::ios_base>()
    struct get_value_type<T> { using type = typename T::char_type; };

template<typename T>
  using value_type = typename get_value_type<Strip<T>>::type;
} // end namespace core_impl

// Value type
template<typename T>
  using Value_type = core_impl::value_type<T>;
```

C++17协程

```
#include <experimental/generator>
#include <iostream>
std::experimental::generator<int> fib(int n) {
    int a = 0;
    int b = 1;
    while (n-- > 0) {
        yield a;
        auto next = a + b;
        a = b;
        b = next;
    }
}

int main() {
    for (auto i : fib(10)) {
        std::cout << i << " ";
    }
}
```

C++17协程

- Output:

0 1 1 2 3 5 8 13 21 34

编译器：VC++ 2015

编译选项：/await