Modern C++元编程应用

祁 宇
qicosmos@163.com

purecpp.org

模版元

```
    // Helpers
    /// \name           Helpers
    //@{
    public:
        template <class Template, class Function, class Tuple, typename... Kinds, class = typename std::enable_if<(sizeof...(Kinds)+1 <= std::tuple_size<Template>::value)>::type> static constexpr Templa
        template <class Template, class Function, class Tuple, typename... Kinds, class = typename std::enable_if<(sizeof...(Kinds) == std::tuple_size<Template>::value)>::type> static constexpr Template
        template <class Template, class Function, typename Kind, class = typename std::enable_if<std::is_convertible<Template, int>::value>::type> static constexpr Template apply(Function&& f, Kind&& va
        template <typename Kind, class Operation = std::plus<Kind>, unsigned int Index = 0, class Tuple, bool Condition = (Index+2 <= std::tuple_size<typename std::remove_cv<typename std::remove_referen
        template <typename Kind, class Operation = std::plus<Kind>, typename... Kinds, class = typename std::enable_if<(std::is_convertible<Kind, int>::value) && (std::is_convertible<typename std::resul
        template <typename Kind, class Operation = std::plus<Kind>, class = typename std::enable_if<(std::is_convertible<Kind, int>::value) && (std::is_convertible<typename std::result_of<Operation(Kind
        template <typename Kind, int Exponent = 1, int One = 1, bool Greater = (Exponent > 1), bool Less = (Exponent < 0), bool Equal = (Exponent == 1), class = typename std::enable_if<std::is_convertib
        template <typename Integer, Integer Zero = Integer(), Integer One = Integer(1), Integer Ones = ~Zero, Integer Size = sizeof(Integer)*std::numeric_limits<unsigned char>::digits, class = typename
        template <typename Integer, Integer Index = Integer(), Integer Zero = Integer(), Integer One = Integer(1), Integer Condition = (Index+One <= sizeof(Integer)*std::numeric_limits<unsigned char>::d
        template <typename Integer, Integer Index = Integer(), Integer Zero = Integer(), Integer One = Integer(1), Integer Size = sizeof(Integer)*std::numeric_limits<unsigned char>::digits, Integer Cond
        template <typename Integer, Integer Mask = ~Integer(), Integer Step = Integer(1), Integer One = Integer(1), Integer Ones = ~Integer(), Integer Condition = (Step+One <= sizeof(Integer)*std::numer
        template <typename Integer, Integer Mask = ~Integer(), Integer Step = Integer(1), Integer One = Integer(1), Integer Ones = ~Integer(), Integer Condition = (Step+One <= nhp<Integer>(sizeof(Intege
        template <typename Integer, Integer Mask = ~Integer(), Integer Step = nhp<Integer>(sizeof(Integer)*std::numeric_limits<unsigned char>::digits), Integer Zero = Integer(), Integer One = Integer(1
        template <typename Integer, Integer Mask = ~Integer(), Integer Step = nhp<Integer>(sizeof(Integer)*std::numeric_limits<unsigned char>::digits), Integer Zero = Integer(), Integer One = Integer(1
        template <typename Integer, Integer Mask = ~Integer(), Integer Step = Integer(), Integer One = Integer(1), Integer Condition = (Step+One <= sizeof(Integer)*std::numeric_limits<unsigned char>::di
        template <typename Integer, Integer Mask = ~Integer(), Integer Step = Integer(), Integer Shift = Integer(), Integer One = Integer(1), Integer Condition = (Step+One <= sizeof(Integer)*std::numer
        template <typename Integer, Integer Mask = ~Integer(), Integer Step = Integer(), Integer Shift = Integer(), Integer One = Integer(1), Integer Condition = (Step+One <= sizeof(Integer)*std::numer
        template <typename Integer, Integer Mask = ~Integer(), Integer Period = sizeof(Integer)*std::numeric_limits<unsigned char>::digits, Integer Step = Integer(), Integer Count = Integer(), Integer
        template <typename Integer, Integer Mask = ~Integer(), Integer Period = sizeof(Integer)*std::numeric_limits<unsigned char>::digits, Integer Step = Integer(), Integer Count = Integer(), Integer
        template <typename Integer, Integer Mask = ~Integer(), Integer Length = sizeof(Integer)*std::numeric_limits<unsigned char>::digits, bool Msb = false, std::size_t Step = Integer(), Integer Zero =
        template <typename Integer, Integer Mask = ~Integer(), Integer Length = sizeof(Integer)*std::numeric_limits<unsigned char>::digits, bool Msb = false, std::size_t Step = Integer(), Integer Zero =
        template <typename Integer, Integer Mask = ~Integer(), Integer Length = sizeof(Integer)*std::numeric_limits<unsigned char>::digits, bool Msb = false, std::size_t Step = Integer(), Integer Zero =
    //@}
```

```
type, Kind>::value) && (std::tuple_size<typename std::remove_cv<typename std::remove_reference<Tuple>::type>::type>::value >= 1)>::type> static constexpr Kind accumulate(Tuple&& tuple);
template <typename Kind, class Operation = std::plus<Kind>, typename... Kinds, class = typename std::enable_if<(std::is_convertible<Kind, int>::value) && (std::is_convertible<typename std::result_of<Operation(Kind, Kind)>::type, Kind>::value)>::type> static constexpr Kind accumulate(const Kind value, const Kinds... values);
template <typename Kind, class Operation = std::plus<Kind>, class = typename std::enable_if<(std::is_convertible<Kind, int>::value) && (std::is_convertible<typename std::result_of<Operation(Kind, Kind)>::type, Kind>::value)>::type> static constexpr Kind accumulate(const Kind value);
template <typename Kind, int Exponent = 1, int One = 1, bool Greater = (Exponent > 1), bool Less = (Exponent < 0), bool Equal = (Exponent == 1), class = typename std::enable_if<std::is_convertible<Kind, int>::value>::type> static constexpr Kind pow(const Kind value);
template <typename Integer, Integer Zero = Integer(), Integer One = Integer(1), Integer Ones = ~Zero, Integer Size = sizeof(Integer)*std::numeric_limits<unsigned char>::digits, class = typename std::enable_if<((std::is_integral<Integer>::value) ? (std::is_unsigned<Integer>::value) : (std::is_convertible<Integer, int>::value)) && (!std::is_floating_point<Integer>::value)>::type> static constexpr Integer block(const Integer location = Zero, const Integer length = Size);
template <typename Integer, Integer Index = Integer(), Integer Zero = Integer(), Integer One = Integer(1), Integer Condition = (Index+One <= sizeof(Integer)*std::numeric_limits<unsigned char>::digits), class = typename std::enable_if<((std::is_integral<Integer>::value) ? (std::is_unsigned<Integer>::value) : (std::is_convertible<Integer, int>::value)) && (!std::is_floating_point<Integer>::value)>::type> static constexpr Integer periodic(const Integer period = One, const Integer offset = Zero);
template <typename Integer, Integer Index = Integer(), Integer Zero = Integer(), Integer One = Integer(1), Integer Size = sizeof(Integer)*std::numeric_limits<unsigned char>::digits, Integer Condition = (Index+One <= Size), class = typename std::enable_if<((std::is_integral<Integer>::value) ? (std::is_unsigned<Integer>::value) : (std::is_convertible<Integer, int>::value)) && (!std::is_floating_point<Integer>::value)>::type> static constexpr Integer comb(const Integer period = One, const Integer offset = Zero);
template <typename Integer, Integer Mask = ~Integer(), Integer Step = Integer(1), Integer One = Integer(1), Integer Ones = ~Integer(), Integer Condition = (Step+One <= sizeof(Integer)*std::numeric_limits<unsigned char>::digits), class = typename std::enable_if<((std::is_integral<Integer>::value) ? (std::is_unsigned<Integer>::value) : (std::is_convertible<Integer, int>::value)) && (!std::is_floating_point<Integer>::value)>::type> static constexpr Integer nhp(const Integer value);
template <typename Integer, Integer Mask = ~Integer(), Integer Step = Integer(1), Integer One = Integer(1), Integer Ones = ~Integer(), Integer Condition = (Step+One <= nhp<Integer>(sizeof(Integer)*std::numeric_limits<unsigned char>::digits)), class = typename std::enable_if<((std::is_integral<Integer>::value) ? (std::is_unsigned<Integer>::value) : (std::is_convertible<Integer, int>::value)) && (!std::is_floating_point<Integer>::value)>::type> static constexpr Integer bhsmask(const Integer value);
template <typename Integer, Integer Mask = ~Integer(), Integer Step = nhp<Integer>(sizeof(Integer)*std::numeric_limits<unsigned char>::digits), Integer Zero = Integer(), Integer One = Integer(1), Integer Ones = ~Zero, Integer Size = nhp<Integer>(sizeof(Integer)*std::numeric_limits<unsigned char>::digits), Integer Temporary = block<Integer>(Step, Step), class = typename std::enable_if<((std::is_integral<Integer>::value) ? (std::is_unsigned<Integer>::value) : (std::is_convertible<Integer, int>::value)) && (!std::is_floating_point<Integer>::value)>::type> static constexpr Integer lzcnt(const Integer value);
template <typename Integer, Integer Mask = ~Integer(), Integer Step = nhp<Integer>(sizeof(Integer)*std::numeric_limits<unsigned char>::digits), Integer Zero = Integer(), Integer One = Integer(1), Integer Ones = ~Zero, Integer Size = nhp<Integer>(sizeof(Integer)*std::numeric_limits<unsigned char>::digits), Integer Temporary = periodic<Integer>(Step*(One+One)+(Step == Zero)), class = typename std::enable_if<((std::is_integral<Integer>::value) ? (std::is_unsigned<Integer>::value) : (std::is_convertible<Integer, int>::value)) && (!std::is_floating_point<Integer>::value)>::type> static constexpr Integer tzcnt(const Integer value);
template <typename Integer, Integer Mask = ~Integer(), Integer Step = Integer(), Integer One = Integer(1), Integer Condition = (Step+One <= sizeof(Integer)*std::numeric_limits<unsigned char>::digits), Integer Temporary = ((Condition) ? ((Mask >> Step) & (Condition)) : (Condition)), class = typename std::enable_if<((std::is_integral<Integer>::value) ? (std::is_unsigned<Integer>::value) : (std::is_convertible<Integer, int>::value)) && (!std::is_floating_point<Integer>::value)>::type> static constexpr Integer popcnt(const Integer value);
template <typename Integer, Integer Mask = ~Integer(), Integer Step = Integer(), Integer Shift = Integer(), Integer One = Integer(1), Integer Condition = (Step+One <= sizeof(Integer)*std::numeric_limits<unsigned char>::digits), Integer Temporary = ((Condition) ? ((Mask >> Step) & (Condition)) : (Condition)), class = typename std::enable_if<((std::is_integral<Integer>::value) ? (std::is_unsigned<Integer>::value) : (std::is_convertible<Integer, int>::value)) && (!std::is_floating_point<Integer>::value)>::type> static constexpr Integer pext(const Integer value);
template <typename Integer, Integer Mask = ~Integer(), Integer Step = Integer(), Integer Shift = Integer(), Integer One = Integer(1), Integer Condition = (Step+One <= sizeof(Integer)*std::numeric_limits<unsigned char>::digits), Integer Temporary = ((Condition) ? ((Mask >> Step) & (Condition)) : (Condition)), class = typename std::enable_if<((std::is_integral<Integer>::value) ? (std::is_unsigned<Integer>::value) : (std::is_convertible<Integer, int>::value)) && (!std::is_floating_point<Integer>::value)>::type> static constexpr Integer pdep(const Integer value);
template <typename Integer, Integer Mask = ~Integer(), Integer Period = sizeof(Integer)*std::numeric_limits<unsigned char>::digits, Integer Step = Integer(), Integer Count = Integer(), Integer Zero = Integer(), Integer One = Integer(1), Integer Size = sizeof(Integer)*std::numeric_limits<unsigned char>::digits, Integer Condition = (Step+One <= Size), Integer Population = popcnt<Integer>(Mask)+((Period-(popcnt<Integer>(Mask)%Period))*(popcnt<Integer>(Mask)%Period != Zero)), Integer Destination = (((((Condition) ? ((Mask >> Step) & (Condition)) : (Condition)) ? (Count) : (Zero))%((Population+(Period*(Population+One <= Period)))/Period))*Period)+((((Condition) ? ((Mask >> Step) & (Condition)) : (Condition)) ? (Count) : (Zero))/((Population+(Period*(Population+One <= Period)))/Period)), Integer Temporary = ((Condition) ? ((Mask >> Step) & (Condition)) : (Condition)) && (Destination < Size), class = typename std::enable_if<((std::is_integral<Integer>::value) ? (std::is_unsigned<Integer>::value) : (std::is_convertible<Integer, int>::value)) && (!std::is_floating_point<Integer>::value)>::type> static constexpr Integer itlc(const Integer value);
template <typename Integer, Integer Mask = ~Integer(), Integer Period = sizeof(Integer)*std::numeric_limits<unsigned char>::digits, Integer Step = Integer(), Integer Count = Integer(), Integer Zero = Integer(), Integer One = Integer(1), Integer Size = sizeof(Integer)*std::numeric_limits<unsigned char>::digits, Integer Condition = (Step+One <= Size), Integer Population = popcnt<Integer>(Mask)+((Period-(popcnt<Integer>(Mask)%Period))*(popcnt<Integer>(Mask)%Period != Zero)), Integer Destination = (((((Condition) ? ((Mask >> Step) & (Condition)) : (Condition)) ? (Count) : (Zero))%Period)*(Population/Period))+((((Condition) ? ((Mask >> Step) & (Condition)) : (Condition)) ? (Count) : (Zero))/Period), Integer Temporary = ((Condition) ? ((Mask >> Step) & (Condition)) : (Condition)) && (Destination < Size), class = typename std::enable_if<((std::is_integral<Integer>::value) ? (std::is_unsigned<Integer>::value) : (std::is_convertible<Integer, int>::value)) && (!std::is_floating_point<Integer>::value)>::type> static constexpr Integer dtlc(const Integer value);
template <typename Integer, Integer Mask = ~Integer(), Integer Length = sizeof(Integer)*std::numeric_limits<unsigned char>::digits, bool Msb = false, std::size_t Step = Integer(), Integer Zero = Integer(), Integer Direction = ((!Msb) || ((Length*(Step+1)) <= (sizeof(Integer)*std::numeric_limits<unsigned char>::digits))), Integer Left = ((Msb) ? (sizeof(Integer)*std::numeric_limits<unsigned char>::digits-(Length*(Step+1))) : (Length*Step))*(Direction), Integer Right = ((Msb) ? ((Length*(Step+1))-sizeof(Integer)*std::numeric_limits<unsigned char>::digits) : (Length*Step))*(!Direction), Integer Condition = ((Left+1 <= sizeof(Integer)*std::numeric_limits<unsigned char>::digits) && (Right+1 <= sizeof(Integer)*std::numeric_limits<unsigned char>::digits) && (Right+1 <= Length)), class Tuple, Integer Count = ((std::tuple_size<typename std::remove_cv<typename std::remove_reference<Tuple>::type>::type>::value)-(Step+1)), class = typename std::enable_if<((std::is_integral<Integer>::value) ? (std::is_unsigned<Integer>::value) : (std::is_convertible<Integer, int>::value)) && (!std::is_floating_point<Integer>::value) && (std::is_convertible<typename std::tuple_element<Step, typename std::remove_cv<typename std::remove_reference<Tuple>::type>::type>::type, Integer>::value)>::type> static constexpr Integer glue(
```

```
view<char>]'
/home/feather/cinatra/main.cpp:183:3:   required from here
/usr/include/c++/7/bits/std_function.h:541:2: error: no type named ‘type’ in ‘class std::result_of<std::_Bind<void (cinatra::http_router::*(cinatra::http_router*, std::_Placeholder<1>, std::_Placeholder<2>,
 main()::<lambda(cinatra::request&, cinatra::response&)>))(cinatra::request&, cinatra::response&, main()::<lambda(cinatra::request&, cinatra::response&)>)>&(const cinatra::request&, cinatra::response&)>’
/home/feather/cinatra/http_router.hpp: In instantiation of ‘void cinatra::http_router::register_nonmember_func(std::string_view, const string&, Function, AP&& ...) [with Function = main()::<lambda(cinatra::re
quest&, cinatra::response&)>; AP = {}; std::string_view = std::basic_string_view<char>; std::__cxx11::string = std::__cxx11::basic_string<char>]’ :
/home/feather/cinatra/http_router.hpp:32:6:   required from ‘ std::enable_if_t<(! is_member_function_pointer_v<Function>)> cinatra::http_router::register_handler(std::string_view, Function&&, Ap&& ...) [with c
inatra::http_method ...Is = {(cinatra::http_method)1, (cinatra::http_method)3}; Function = main()::<lambda(cinatra::request&, cinatra::response&)>; Ap = {}; std::enable_if_t<(! is_member_function_pointer_v<Fun
ction>)> = void; std::string_view = std::basic_string_view<char>]’
/home/feather/cinatra/http_server.hpp:185:5:   required from ‘ void cinatra::http_server_<service_pool_policy>::set_http_handler(std::string_view, Function&&, AP&& ...) [with cinatra::http_method ...Is = {(cin
atra::http_method)1, (cinatra::http_method)3}; Function = main()::<lambda(cinatra::request&, cinatra::response&)>; AP = {}; service_pool_policy = cinatra::io_service_pool; std::string_view = std::basic_string_
view<char>]’
/home/feather/cinatra/main.cpp:183:3:   required from here
/usr/include/c++/7/bits/std_function.h:550:2: note: candidate: template<class _Functor> std::function<_Res(_ArgTypes ...)>& std::function<_Res(_ArgTypes ...)>::operator=(std::reference_wrapper<_Functor>) [with
 _Functor = _Functor; _Res = void; _ArgTypes = {const cinatra::request&, cinatra::response&}]
  operator=(reference_wrapper<_Functor> __f) noexcept
  ^~~~~~~~
/usr/include/c++/7/bits/std_function.h:550:2: note:   template argument deduction/substitution failed:
In file included from /home/feather/cinatra/http_server.hpp:16:0,
                 from /home/feather/cinatra/main.cpp:2:
/home/feather/cinatra/http_router.hpp:113:31: note:   ‘ std::_Bind<void (cinatra::http_router::*(cinatra::http_router*, std::_Placeholder<1>, std::_Placeholder<2>, main()::<lambda(cinatra::request&, cinatra::r
esponse&)>))(cinatra::request&, cinatra::response&, main()::<lambda(cinatra::request&, cinatra::response&)>)>’ is not derived from ‘ std::reference_wrapper< Tp>’
/usr/include/c++/7/bits/std_function.h:480:7: note: candidate: std::function<_Res(_ArgTypes ...)>& std::function<_Res(_ArgTypes ...)>::operator=(const std::function<_Res(_ArgTypes ...)>&) [with _Res = void; _A
rgTypes = {const cinatra::request&, cinatra::response&}]
       operator=(const function& __x)
       ^~~~~~~~
/usr/include/c++/7/bits/std_function.h:480:7: note:   no known conversion for argument 1 from ‘ std::_Bind_helper<false, void (cinatra::http_router::*)(cinatra::request&, cinatra::response&, main()::<lambda(ci
natra::request&, cinatra::response&)>, check, log_t), cinatra::http_router*, const std::_Placeholder<1>&, const std::_Placeholder<2>&, main()::<lambda(cinatra::request&, cinatra::response&)>, check, log_t>::ty
pe {aka std::_Bind<void (cinatra::http_router::*(cinatra::http_router*, std::_Placeholder<1>, std::_Placeholder<2>, main()::<lambda(cinatra::request&, cinatra::response&)>, check, log_t))(cinatra::request&, ci
natra::response&, main()::<lambda(cinatra::request&, cinatra::response&)>, check, log_t)>}’ to ‘ const std::function<void(const cinatra::request&, cinatra::response&)>&’
/usr/include/c++/7/bits/std_function.h:498:7: note: candidate: std::function<_Res(_ArgTypes ...)>& std::function<_Res(_ArgTypes ...)>::operator=(std::function<_Res(_ArgTypes ...)>&&) [with _Res = void; _ArgTyp
es = {const cinatra::request&, cinatra::response&}]
       operator=(function&& __x) noexcept
       ^~~~~~~~
/usr/include/c++/7/bits/std_function.h:498:7: note:   no known conversion for argument 1 from ‘ std::_Bind_helper<false, void (cinatra::http_router::*)(cinatra::request&, cinatra::response&, main()::<lambda(ci
natra::request&, cinatra::response&)>, check, log_t), cinatra::http_router*, const std::_Placeholder<1>&, const std::_Placeholder<2>&, main()::<lambda(cinatra::request&, cinatra::response&)>, check, log_t>::ty
pe {aka std::_Bind<void (cinatra::http_router::*(cinatra::http_router*, std::_Placeholder<1>, std::_Placeholder<2>, main()::<lambda(cinatra::request&, cinatra::response&)>, check, log_t))(cinatra::request&, ci
natra::response&, main()::<lambda(cinatra::request&, cinatra::response&)>, check, log_t)>}’ to ‘ std::function<void(const cinatra::request&, cinatra::response&)>&&’
/usr/include/c++/7/bits/std_function.h:512:7: note: candidate: std::function<_Res(_ArgTypes ...)>& std::function<_Res(_ArgTypes ...)>::operator=(std::nullptr_t) [with _Res = void; _ArgTypes = {const cinatra::r
equest&, cinatra::response&}; std::nullptr_t = std::nullptr_t]
       operator=(nullptr_t) noexcept
       ^~~~~~~~
/usr/include/c++/7/bits/std_function.h:512:7: note:   no known conversion for argument 1 from ‘ std::_Bind_helper<false, void (cinatra::http_router::*)(cinatra::request&, cinatra::response&, main()::<lambda(ci
natra::request&, cinatra::response&)>, check, log_t), cinatra::http_router*, const std::_Placeholder<1>&, const std::_Placeholder<2>&, main()::<lambda(cinatra::request&, cinatra::response&)>, check, log_t>::ty
pe {aka std::_Bind<void (cinatra::http_router::*(cinatra::http_router*, std::_Placeholder<1>, std::_Placeholder<2>, main()::<lambda(cinatra::request&, cinatra::response&)>, check, log_t))(cinatra::request&, ci
natra::response&, main()::<lambda(cinatra::request&, cinatra::response&)>, check, log_t)>}’ to ‘ std::nullptr_t’
/usr/include/c++/7/bits/std_function.h:541:2: note: candidate: template<class _Functor> std::function<_Res(_ArgTypes ...)>::_Requires<std::function<_Res(_ArgTypes ...)>::_Callable<typename std::decay<_U1>::typ
e>, std::function<_Res(_ArgTypes ...)>&> std::function<_Res(_ArgTypes ...)>::operator=(_Functor&&) [with _Functor = _Functor; _Res = void; _ArgTypes = {const cinatra::request&, cinatra::response&}]
  operator=(_Functor&& __f)
  ^~~~~~~~
/usr/include/c++/7/bits/std_function.h:541:2: note:   template argument deduction/substitution failed:
/usr/include/c++/7/bits/std_function.h: In substitution of ‘ template<class _Functor> std::function<void(const cinatra::request&, cinatra::response&)>::_Requires<std::function<void(const cinatra::request&, cin
atra::response&)>::_Callable<typename std::decay<_Tp>::type, typename std::result_of<typename std::decay<_Tp>::type&(const cinatra::request&, cinatra::response&)>::type>, std::function<void(const cinatra::requ
est&, cinatra::response&)>&> std::function<void(const cinatra::request&, cinatra::response&)>::operator=<_Functor>(_Functor&&) [with _Functor = std::_Bind<void (cinatra::http_router::*(cinatra::http_router*, s
td::_Placeholder<1>, std::_Placeholder<2>, main()::<lambda(cinatra::request&, cinatra::response&)>, check, log_t))(cinatra::request&, cinatra::response&, main()::<lambda(cinatra::request&, cinatra::response&)>
, check, log_t)>]’ :
```


!! 模版元
代码墙!!
在中国C++大会上??

元编程的缺点

- 代码晦涩
- 比较难写
- 编译时间长
- 糟糕的错误提示

元编程的优点

- zero-overhead 编译期计算
- 简洁而优雅地解决问题
- 终极抽象

Dream code!

Meta Programming编程思想

- In C++98
  - 元函数
  - SFINAE
  - 模版递归
  - 递归继承
  - Tag Dispatch
  - 模版特化/偏特化

Meta Programming编程思想

- In C++98

```
template<class T>
struct add_pointer { typedef T* type; };

typedef typename add_pointer<int>::type int_pointer;
```

元函数：编译期函数调用的类或模版类– add_pointer

调用元函数：访问元函数的内部类型::type

Type T作为元函数的value，类型是元编程中的一等公民

模版元编程概念上是函数式编程

Meta Programming编程思想

- In C++98

```
template<bool B, class T = void>
struct enable_if {};

template<class T>
struct enable_if<true, T> { typedef T type; };
```

```
template <class T>
typename enable_if<sizeof(T)==4, void>::type
foo(T  t) {}

foo(1);  //ok
foo('a'); //compile error
```

SFINAE：替换失败不是一个错误，基于模版实例化的tag dispatch

Meta Programming编程思想

- In C++98

```
template <int n> struct fact98 {
   static const int value = n * fact98<n - 1>::value;
};
template <> struct fact98<0> {
   static const int value = 1;
};
std::cout << fact98<5>::value << std::endl;
```

模版递归

Meta Programming编程思想

- In C++98

  模版元编程集大成者：boost.mpl, boost.fusion

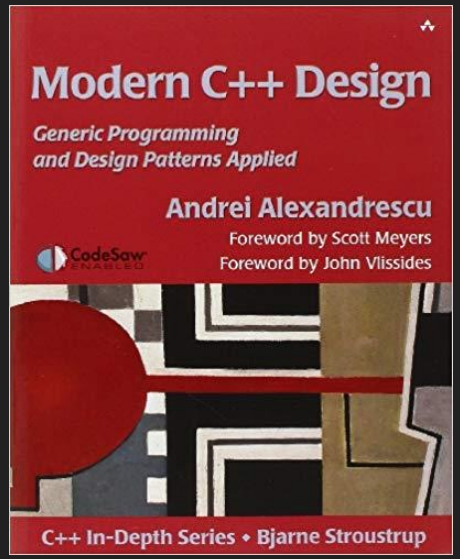
  boost.mpl：编译期类型容器和算法

  boost.fusion：通过异构的编译期容器融合编译期和运行期计算

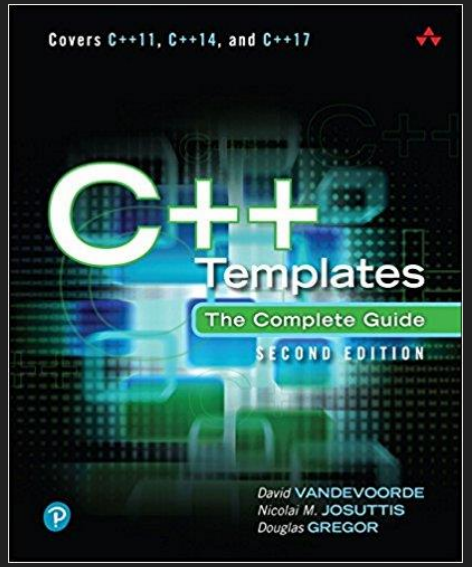
```
struct print{
template <typename T>
   void operator()(T const& x) const{
      std::cout << typeid(x).name() << std::endl;
   }
};

template <typename Sequence>
void print_names(Sequence const& seq){
   for_each(filter_if<boost::is_class<_> >(seq), print());
}
boost::fusion::vector<int, char, std::string> stuff(2018, 'i', "purecpp");
print_names(stuff);
```

Meta Programming编程思想

- In C++98



Andrei Alexandrescu



Meta Programming编程思想

- In C++98

```
template<class T>
struct add_pointer { typedef T* type; };

typedef typename add_pointer<int>::type int_pointer;
```

- In C++11

```
template<class T> using add_pointer = T*;

using  int_pointer = add_pointer<int>;
```

C++11 template aliases（模版别名）

元函数由类和类模版变为模版别名

Meta Programming编程思想

- In C++11

```
template<typename... Values> struct meta_list {};
using list_of_ints = meta_list<char, short, int, long>;

template<class List> struct list_size;
template<template<class...> class List, class... Elements>
struct list_size<List<Elements...>>
   : std::integral_constant<std::size_t, sizeof...(Elements)> {};

constexpr auto size = list_size<std::tuple<int, float, void>>::value;
constexpr auto size1 = list_size<list_of_ints>::value;
constexpr auto size2 = list_size<boost::variant<int, float>>::value;
```

C++11 Variadic Templates

Variadic templates作为类型容器

通过variadic templates pack访问模版参数，不通过模版递归和特化

Meta Programming编程思想

is_void (C++11)	checks if a type is `void` (class template)
is_null_pointer (C++14)	checks if a type is `std::nullptr_t` (class template)
is_integral (C++11)	checks if a type is integral type (class template)
is_floating_point (C++11)	checks if a type is floating-point type (class template)
is_array (C++11)	checks if a type is an array type (class template)
is_enum (C++11)	checks if a type is an enumeration type (class template)
is_union (C++11)	checks if a type is an union type (class template)
is_class (C++11)	checks if a type is a non-union class type (class template)
is_function (C++11)	checks if a type is a function type (class template)
is_pointer (C++11)	checks if a type is a pointer type (class template)
is_lvalue_reference (C++11)	checks if a type is *lvalue reference* (class template)
is_rvalue_reference (C++11)	checks if a type is *rvalue reference* (class template)
is_member_object_pointer (C++11)	checks if a type is a pointer to a non-static member object (class template)
is_member_function_pointer (C++11)	checks if a type is a pointer to a non-static member function (class template)

C++11 type_traits**提供了大量的元函数**，让模版元编程变得更简单

Meta Programming编程思想

- In C++98

```
template <int n> struct fact98 {
   static const int value = n * fact98<n - 1>::value;
};
template <> struct fact98<0> {
   static const int value = 1;
};
std::cout << fact98<5>::value << std::endl;
```

- In C++11

```
constexpr int fact11(int n) {
   return n <= 1 ? 1 : (n * fact11(n - 1));
}
```

Meta Programming编程思想

- In C++11

```
constexpr int fact11(int n) {
   return n <= 1 ? 1 : (n * fact11(n - 1));
}
```

- In C++14

```
constexpr  int  fact14(int n) {
  int s = 1;
  for (int  i = 1; i <= n; i++) { s = s * i; }
  return s;
}
```

Meta Programming编程思想

- In C++14

Everything changed in meta programming

新特性产生新的编程思想！

constexpr, generic lambda, variable template

Louis Dionne

Boost.hana

Meta Programming编程思想

- Boost.hana

```
template <typename T>
struct type_wrapper {
    using type = T;
};
```

```
template <typename T>
type_wrapper<T> type{};
```

```
//type to value
auto the_int_type = type<int>;
```

```
//value to type
using the_real_int_type = decltype(the_int_type)::type;
```

Meta Programming编程思想

- Boost.hana

```
template <typename T>
type_wrapper<T> type{};

constexpr auto add_pointer = [](auto t) {
    using T = typename decltype(t)::type;
    return type<std::add_pointer_t<T>>; //type to value
};
```

```
constexpr auto intptr = add_pointer(type<int>);
static_assert(std::is_same_v<decltype(intptr)::type, int*>); //value to type
```

元函数的定义不再是类了，而是lambda!

generic lambda, variable template, constexpr

functional programming

Meta Programming编程思想

- Boost.hana

```
auto animal_types = hana::make_tuple(hana::type_c<Fish*>, hana::type_c<Cat&>, hana::type_c<Dog*>);
auto animal_ptrs = hana::filter(animal_types, [](auto a) {
      return hana::traits::is_pointer(a);
});

static_assert(animal_ptrs == hana::make_tuple(hana::type_c<Fish*>, hana::type_c<Dog*>), "");
```

```
auto animals = hana::make_tuple(Fish{ "Nemo" }, Cat{ "Garfield" }, Dog{ "Snoopy" });
auto names = hana::transform(animals, [](auto a) {
      return a.name;
});
assert(hana::reverse(names) == hana::make_tuple("Snoopy", "Garfield", "Nemo"));
```

通过类型容器融合编译期和运行期计算，替代boost.mpl和boost.fusion!

Meta Programming编程思想

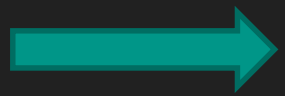
- Boost.hana
  - **元函数不再是**类或类模版，而是lambda
  - **不再基于**类型，而是基于值
  - **没有**SFINAE，**没有模版**递归
  - **函数式**编程
  - **代**码更容易理解
  - **元**编程变得更简单
  - **融合**编译期与运行期

Meta Programming编程思想

- In C++14

```
template<std::size_t I>
auto& get(person& p);

template<>
auto& get<0>(person& p) {
    return p.id;
}

template<>
auto& get<1>(person& p) {
    return p.name;
}

template<>
auto& get<2>(person& p) {
    return p.age;
}
```

In C++17



```
template<std::size_t I>
auto& get(person& p) {
    if constexpr (I == 0) {
        return p.id;
    }
    else if constexpr (I == 1) {
        return p.name;
    }
    else if constexpr (I == 2) {
        return p.age;
    }
}
```

没有特化

Meta Programming编程思想

- In C++14

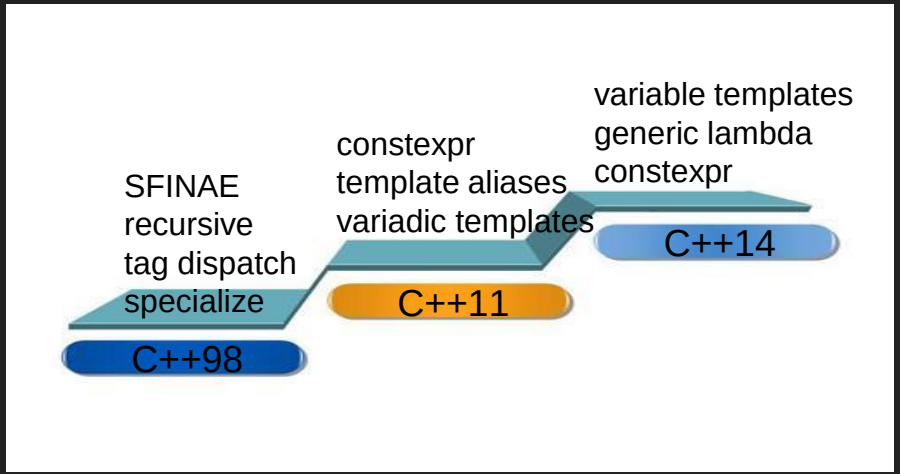
```
template <typename T>
std::enable_if_t<std::is_same_v<T, std::string>, std::string> to_string(T t){
    return t;
}

template <typename T>
std::enable_if_t<!std::is_same_v<T, std::string>, std::string> to_string(T t){
    return std::to_string(t);
}
```

- In C++17

```
template <typename T>
std::string to_string(T t){
    if constexpr(std::is_same_v<T, std::string>)
        return t;
    else
        return std::to_string(t);
}
```

No SFINAE

C++新标准新特性产生新的idea，让元编程变得更简单更强大



Newer is Better！

- C++98: boost.mpl, boost.fusion
- C++11: boost.mp11, meta, brigand
- C++14: boost.hana


Bjarne Stroustrup:
使用最新的 我们正在改进和完善的C++标准

静态检查

静态检查

```
static_assert(sizeof(void *) == 8, "expected 64-bit platform");

template<typename T, int Row, int Column>
struct Matrix {
    static_assert(Row >= 0, "Row number must be positive.");
    static_assert(Row >= 0, "Column number must be positive.");
    static_assert(Row + Column > 0, "Row and Column must be greater than 0.");
};
```

静态检查

```
struct A {
void foo(){}
     int member;
};

template<typename Function>
std::enable_if_t<!std::is_member_function_pointer_v<Function>> foo(Function&& f) {
}

foo([] {});  //ok
foo(&A::foo);  //compile error: no matching function for call to 'foo(void (A::*)())'
```

安全，尽可能早地发现bug，让编译器而不是人帮助发现bug

静态检查——编译期探测

探测

```
template< class, class = void >
struct has_foo : std::false_type {};

template< class T >
struct has_foo< T, std::void_t<decltype(std::declval<T>().foo())> > :
   std::true_type {};
```

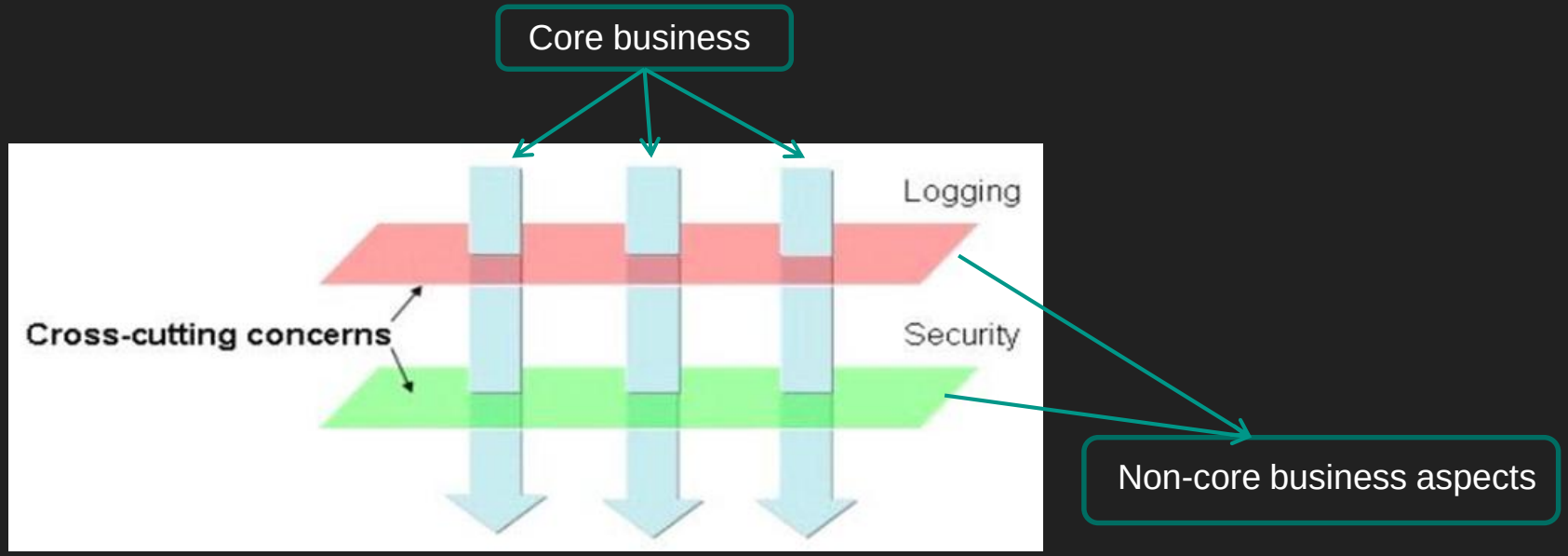
```
template< class, class = void >
struct has_member : std::false_type {};

template< class T >
struct has_member< T, std::void_t<decltype(std::declval<T>().member)> > :
   std::true_type {};
```

```
struct A {
     void foo(){}
     int member;
};
static_assert(has_foo< A >::value);
static_assert(has_member< A >::value);
```

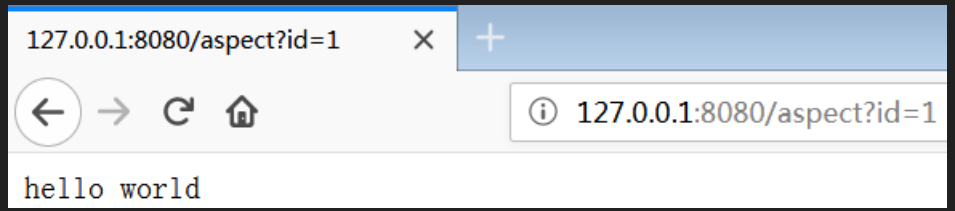
decltype, void_t, SFINAE In C++17

探测

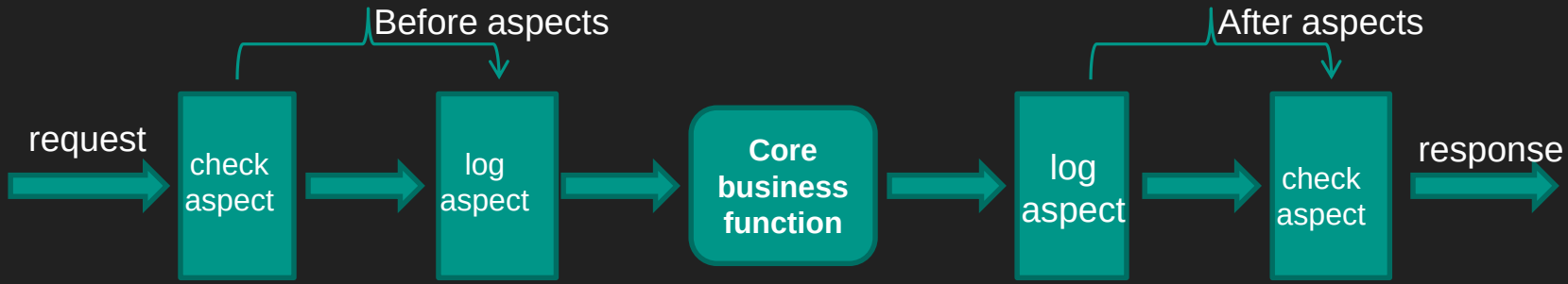
- AOP(Aspect Oriented Programming)



```
server.set_http_handler<GET, POST>("/aspect", [](request& req, response& res) {
    std::cout << "in business function" << std::endl;
    res.render_string("hello world");
}, check{}, log_t{});
```




```
check passed
before log
in business function
after log
after check
```



探测

- AOP

```
constexpr bool has_befor_mtd = has_before<decltype(item), request&, response&>::value;
if constexpr (has_befor_mtd)
   r = item.before(req, res);

constexpr bool has_after_mtd = has_after<decltype(item), request&, response&>::value;
if constexpr (has_after_mtd)
   r = item.after(req, res);
```

探测

- AOP(Aspect Oriented Programming)

```
#define HAS_MEMBER(member)\
template<typename T, typename... Args>\
struct has_##member\
{\
private:\
    template<typename U> static auto Check(int) -> decltype(std::declval<U>().member(std::declval<Args>()...),
std::true_type()); \
    template<typename U> static std::false_type Check(...);\
public:\
    enum{value = std::is_same<decltype(Check<T>(0)), std::true_type>::value};\
};

HAS_MEMBER(before)
HAS_MEMBER(after)
```

AOP in feather : https://github.com/qicosmos/feather

静态检查 -- 编译期探测 -- 编译期计算

编译期计算

- 类型计算
- 类型推导
- 类型萃取
- 类型转换
- 数值计算 **表达式模版**，Xtensor, Eigen, Mshadow

类型萃取

```
template<typename Ret, typename... Args>
struct function_traits_impl<Ret(Args...)>{
public:
     enum { arity = sizeof...(Args) };
     typedef Ret function_type(Args...);
     typedef Ret result_type;
     using stl_function_type = std::function<function_type>;
     typedef Ret(*pointer)(Args...);

     template<size_t I>
     struct args{
     static_assert(I < arity, "index is out of range, index must less than sizeof Args");
          using type = typename std::tuple_element<I, std::tuple<Args...>>::type;
     };

     typedef std::tuple<std::remove_cv_t<std::remove_reference_t<Args>>...> tuple_type;
     using args_type_t = std::tuple<Args...>;
};
```

function_traits in cinatra: https://github.com/qicosmos/cinatra

类型萃取

- rpc路由

```
struct rpc_service {
     int add(int a, int b) { return a + b; }

     std::string translate(const std::string& orignal) {
          std::string temp = orignal;
          for (auto& c : temp) c = toupper(c);
          return temp;
     }
};
```

```
rpc_server server;
server.register_handler("add", &rpc_service::add, &rpc_srv);
server.register_handler("translate", &rpc_service::translate, &rpc_srv);
```

```
auto result = client.call<int>("add", 1, 2);
auto result = client.call<std::string>("translate", "hello");
```

Modern C++ rpc库rest_rpc: https://github.com/qicosmos/rest_rpc

类型萃取

- 路由

```
template<typename Function>
void register_nonmember_func(std::string const& name, const Function& f) {
      this->map_invokers_[name] = { std::bind(&invoker<Function>::apply, f,
std::placeholders::_1,   std::placeholders::_2, std::placeholders::_3) };
}
```

```
template<typename Function>
struct invoker {
      static void apply(const Function& func, const char* data, size_t size,
std::string& result) {
      using args_tuple = typename function_traits<Function>::args_tuple;
      msgpack_codec codec;
      auto tp = codec.unpack<args_tuple>(data, size);
      call(func, result, tp);
   }
};
```

类型萃取

- 路由

```
template<typename F, size_t... I, typename... Args>
typename std::result_of<F(Args...)>::type call_helper(
const F& f, const std::index_sequence<I...>&, const std::tuple<Args...>& tup) {
      return f(std::get<I>(tup)...);
}

template<typename F, typename... Args>
typename std::enable_if<std::is_void<typename
std::result_of<F(Args...)>::type>::value>::type
call(const F& f, std::string& result, std::tuple<Args...>& tp) {
      call_helper(f, std::make_index_sequence<sizeof...(Args)>{}, tp);
      result = msgpack_codec::pack_args_str(result_code::OK);
}
```

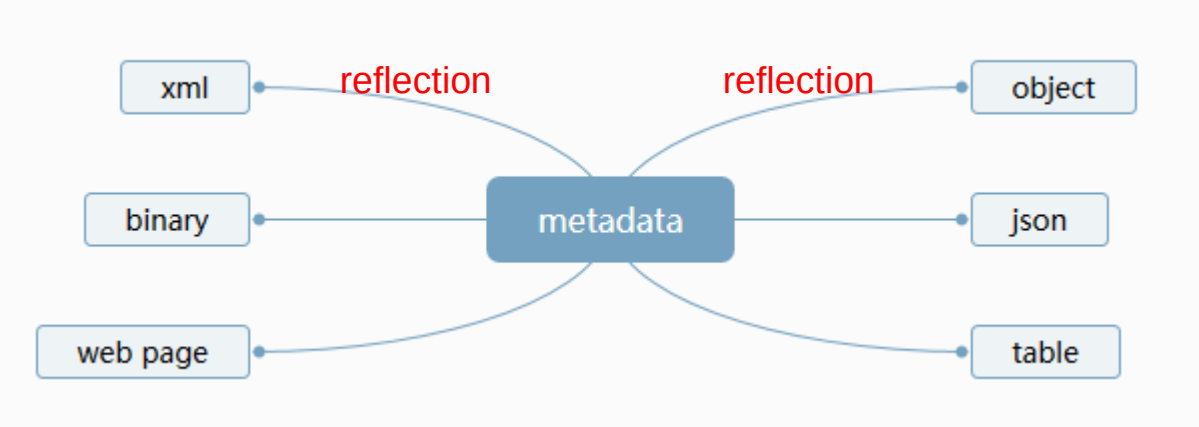
静态检查 -- 编译期探测 -- 编译期计算

编译期反射

编译期反射

- 编译期反射

```
struct person{
   std::string  name;
    int         age;
};
REFLECTION(person, name, age)
```



编译期反射

- 序列化引擎
- ORM
- 协议适配器

```
person p = {"tom", 20};
iguana::string_stream ss;

to_xml(ss, p);
to_json(ss, p);
to_msgpack(ss, p);
to_protobuf(ss, p);
```

```
ormpp::dbng<mysql> mysql;
ormpp:: dbng<sqlite> sqlite;
ormpp:: dbng<postgresql> postgres;

mysql.create_datatable<person>()
sqlite.create_datatable<person>()
postgres.create_datatable<person>()
```

基于编译期反射的序列化引擎iguana: https://github.com/qicosmos/iguana

基于编译期反射的ORM库ormpp: https://github.com/qicosmos/ormpp

静态检查 -- 编译期探测 -- 编译期计算

编译期反射

融合编译期和运行期

融合编译期和运行期

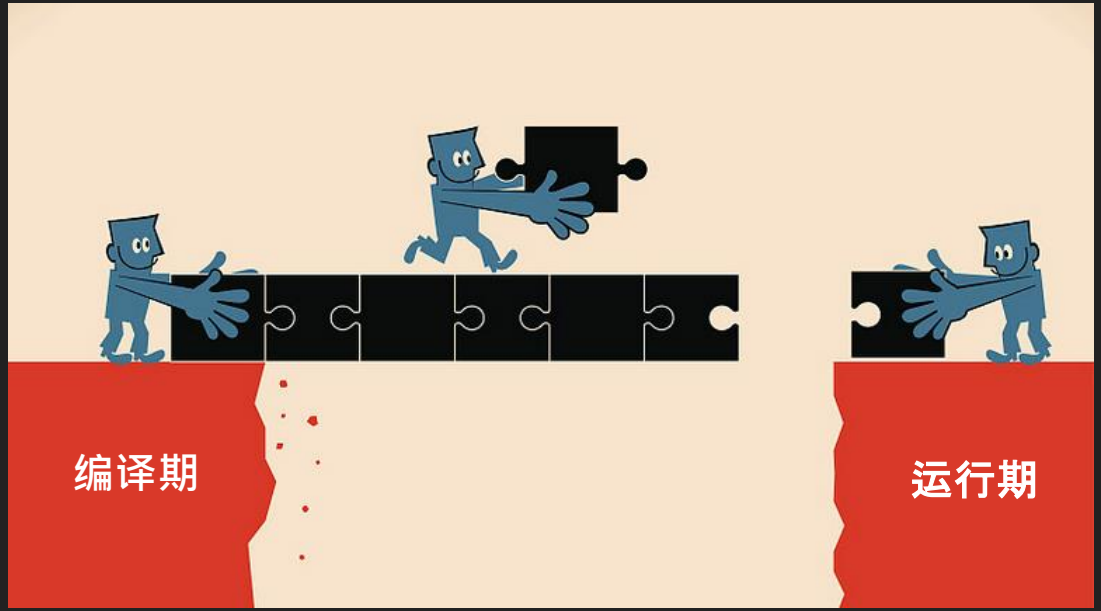
- 从运行期到编译期
  Value to Type

```
auto val = std::integral_constant<int, 5>{};
using int_type = decltype(val);
```

- 从编译期到运行期
  Type to Value

```
auto v = decltype(val)::value;
```

融合编译期和运行期



融合编译期和运行期

- 如何根据一个运行时的值调用一个编译期模版函数？
- 如何将运行时的网络数据映射为一个函数调用？

融合编译期和运行期

```
template<size_t N>
void fun() {}

void foo(int n) {
     switch (n){
     case 0:
        fun<0>();
        break;
     case 1:
        fun<1>();
        break;
     case 2:
        fun<2>();
        break;
     default:
        break;
    }
}
```

foo(100)??

```
void foo(int n) {
     switch (n){
     case 0:
        fun<0>();
        break;
     case 1:
        fun<1>();
        break;
     case 2:
        fun<2>();
        break;
     ......
     case 99:
        fun<99>();
        break;
     default:
        break;
}
}
```

融合编译期和运行期

```
namespace detail {
    template <class Tuple, class F, std::size_t...Is>
    void tuple_switch(const std::size_t i, Tuple&& t, F&& f, std::index_sequence<Is...>) {
      (void)std::initializer_list<int> {
            (i == Is && (
                  (void)std::forward<F>(f)(std::integral_constant<size_t, Is>{}), 0))...
      };
    }
  } // namespace detail

  template <class Tuple, class F>
  inline void tuple_switch(const std::size_t i, Tuple&& t, F&& f) {
    constexpr auto N =
          std::tuple_size<std::remove_reference_t<Tuple>>::value;

    detail::tuple_switch(i, std::forward<Tuple>(t), std::forward<F>(f),
                  std::make_index_sequence<N>{});
  }
```

融合编译期和运行期

```
void foo(int n) {
   std::tuple<int, int, int> tp;
   tuple_switch(n, tp, [](auto item) {
      constexpr auto I = decltype(item)::value;
      fun<I>();
   });
}
```

```
foo(1);
foo(2);
```

foo(100)??

融合编译期和运行期

```
template<size_t... Is>
auto make_tuple_from_sequence(std::index_sequence<Is...>)->decltype(std::make_tuple(Is...)) {
    std::make_tuple(Is...);
}

template<size_t N>
constexpr auto make_tuple_from_sequence()-
>decltype(make_tuple_from_sequence(std::make_index_sequence<N>{})) {
    return make_tuple_from_sequence(std::make_index_sequence<N>{});
}
```

```
void foo(int n) {
    decltype(make_tuple_from_sequence<100>()) tp;  //std::tuple<int, int, …, int>
    tuple_switch(n, tp, [](auto item) {
        constexpr auto I = decltype(item)::value;
        fun<I>();
    });
}

foo(98);
foo(99);
```

融合编译期和运行期

- std::tuple：编译期和运行期的桥梁
  - 类型容器
  - 异构的值容器
  - 通过遍历方式同时获得值与类型
- std::index_sequence, std::integral_constant
  - 帮助遍历类型和值
  - 帮助获得编译期索引

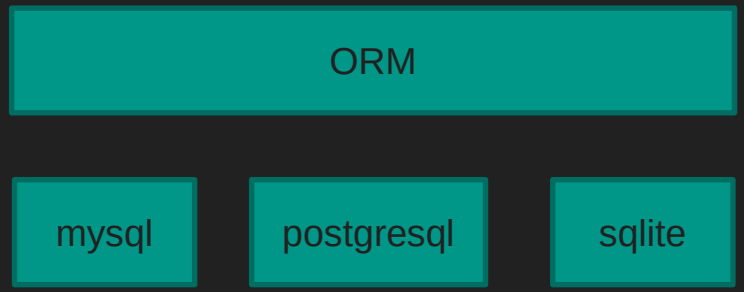
静态检查 — 编译期探测 — 编译期计算

编译期反射 — 融合编译期和运行期

接口泛化与统一

接口泛化与统一

- 融合底层异构的子系统
- 屏蔽差异
- 提供统一的接口



接口泛化与统一

- Mysql connect

```
mysql_real_connect(handle, "127.0.0.1", "feather", "2018", "testdb", 0, nullptr, 0);
```

- Postgresql connect

```
PQconnectdb("host=localhost user=127.0.0.1 password=2018 dbname=testdb");
```

- Sqlite connect

```
sqlite3_open("testdb", handle);
```

- ORM unified connect interface

```
ORM::mysql.connect("127.0.0.1", “feather", “2018", "testdb");
ORM::postgres.connect("127.0.0.1", “feather", “2018", "testdb");
ORM::sqlite.connect("testdb");
```

接口泛化与统一

- 通过可变参数模板统一接口
- 通过policy-base设计和variadic templates来屏蔽数据库接口差异

```
template<typename DB>
class dbng{
template <typename... Args>
        bool connect(Args&&... args){
            return  db_.connect(std::forward<Args>(args)...);
        }
```

```
template<typename... Args>
bool connect(Args... args) {
if constexpr (sizeof...(Args)==5) {
return std::apply(&mysql_real_connect, std::make_tuple(args...);
}
else if constexpr (sizeof...(Args) == 4) {//postgresql}
else if constexpr (sizeof...(Args) == 2) {//sqlite}
}
```

接口泛化与统一

if constexpr + variadic templates = 静态多态

- 通过增加参数或修改参数类型方式来扩展接口
- 没有继承
- 没有SFINAE
- 没有模版特化

Modern C++ ORM库: https://github.com/qicosmos/ormpp

静态检查 — 编译期探测 — 编译期计算

编译期反射

融合编译期和运行期

接口泛化与统一 — 消除重复(宏)

消除重复（宏）

```
#define ENUM_TO_OSTREAM_FUNC(EnumType)                                  \
    std::ostream& operator<<(std::ostream& out_stream, const EnumType& x) { \
      out_stream << static_cast<int>(x);                                \
      return out_stream;                                                \
    }

enum class MsgType { Connected, Timeout };
enum class DataType {Float, Int32};
ENUM_TO_OSTREAM_FUNC(MsgType);
ENUM_TO_OSTREAM_FUNC(DataType);
```

```
std::stringstream ss;
ss << MsgType::Connected<< DataType::Float;
```

消除重复（宏）

```
#define ENUM_TO_OSTREAM_FUNC(EnumType)                                  \
    std::ostream& operator<<(std::ostream& out_stream, const EnumType& x) { \
     out_stream << static_cast<int>(x);                                 \
     return out_stream;                                                 \
    }

enum class MsgType { Connected, Timeout };
enum class DataType {Float, Int32};
ENUM_TO_OSTREAM_FUNC(MsgType);
ENUM_TO_OSTREAM_FUNC(DataType);
```

```
template<typename T, typename = typename
std::enable_if<std::is_enum<T>::value>::type>
std::ostream& operator<<(std::ostream& out_stream, T x) {
    out_stream << static_cast<int>(x);
    return out_stream;
}
```

消除重复（宏）

```
#define CALL(name, ...)                                  \
    do {                                                 \
        result ret = func(name); \
        if (ret == 0) {                    \
              __VA_ARGS__;                                          \
              do_something(name);                \
      }\
       else {                \
              do_something (name);              \
      } \
    } while (0)

CALL("root", func1(root_path));
CALL("temp", func2(temp_path));
```

消除重复（宏）

```
template<typename Self, typename F>
void Call(const std::string& name, Self * self, F f) {
     auto ret = foo(name);
     if (ret == 0) {
          (self >*f)(name);
          do_something(name);
     }
     else {
          do_something(name);
     }
}
```

大部分宏能做的，元编程能做得更好，更完美！

静态检查

编译期探测

编译期计算

编译期反射

融合编译期和运行期

接口泛化与统一

消除重复(宏)

接口易用性和灵活性

接口易用和灵活性

```
struct dummy{
     int add(connection* conn, int a, int b) { return a + b; }
};

int add(connection* conn, int a, int b) { return a + b; }
```

```
rpc_server server(8080, 4);

dummy d;
server.register_handler("a", &dummy::add, &d);
server.register_handler("b", add);
server.register_handler("c", [](connection* conn) {});
server.register_handler("d", [](connection* conn, std::string s) {
     return s;
});
```

同一个接口可以注册任意类型函数（callable）

接口易用和灵活性

- 类型擦除

```
template<typename Function > 擦除
void register_nonmember_func(std::string const& name, const Function& f) {
      this->map_invokers_[name] = { std::bind(&invoker<Function>::apply, f,
std::placeholders::_1,   std::placeholders::_2, std::placeholders::_3) };
}
```

```
template<typename Function>
struct invoker {
      static void apply(const Function& func, const char* data, size_t size,
std::string& result) {
      using args_tuple = typename function_traits<Function>::args_tuple;  还原
      msgpack_codec codec;
      auto tp = codec.unpack<args_tuple>(data, size);
      call(func, result, tp);
   }
};
```

https://github.com/qicosmos/rest_rpc

接口易用和灵活性

```
struct person{
      void foo(request& req, response& res) {
            res.render_string("ok");
      }
};
server.set_http_handler<GET>("/a",  &person::foo);
```

- Play game！

接口易用和灵活性

```
server.set_http_handler<GET>("/a", &person::foo);

server.set_http_handler<GET, POST>("/b", &person::foo, log_t{});

server.set_http_handler<GET, POST, HEAD>("/c", &person::foo, log_t{}, check{});

server.set_http_handler<GET>("/d", &person::foo, log_t{}, check{}, enable_cache{ false });

server.set_http_handler<GET>("/e", &person::foo, log_t{}, enable_cache{ false }, check{});

server.set_http_handler<POST>("/f", &person::foo, enable_cache{ false }, log_t{}, check{});
```

接口易用和灵活性

```
template<http_method... Is, typename Function, typename... AP>
void set_http_handler(std::string_view name, Function&& f, AP&&... ap) {
    if constexpr(has_type<enable_cache<bool>,
std::tuple<std::decay_t<AP>...>>::value) {
        auto tp = filter<enable_cache<bool>>(std::forward<AP>(ap)...);
        std::apply(f, std::move(tp));
    }
    else {
        http_router_.register_handler<Is...>(name, std::forward<Function>(f),
std::forward<AP>(ap)...);
    }
}
```

```
template <typename T, typename Tuple>
struct has_type;

template <typename T, typename... Us>
struct has_type<T, std::tuple<Us...>> : std::disjunction<std::is_same<T, Us>...> {};
```

接口易用和灵活性

```
template< typename T>
struct filter_helper{
     static constexpr auto func(){
          return std::tuple<>();
     }

     template< class... Args >
     static constexpr auto func(T&&, Args&&...args){
          return filter_helper::func(std::forward<Args>(args)...);
     }

     template< class X, class... Args >
     static constexpr auto func(X&&x, Args&&...args){
          return std::tuple_cat(std::make_tuple(std::forward<X>(x)),
filter_helper::func(std::forward<Args>(args)...));
     }
};
```

静态检查

编译期探测

编译期计算

编译期反射

融合编译期和运行期

接口泛化与统一

消除重复(宏)

接口易用性和灵活性

使用元编程

Thank you!

Code: https://github.com/qicosmos
qicosmos@163.com
purecpp.org

Newer is Better!