2017 CPP-Summit

从现代C++元编程到ORM

祁 宇

qicosmos@163.com

purecpp.org

- 新特性（C++11/14/17）
- 元编程
- 元编程的一些用途
- 元编程实现ORM

Modern C++ Creative Ideas

Variadic Template

- 适配不同的接口

```
template<typename... Args>
decltype(auto) interface(Args&&... args);
```

```
struct some_db1{
    bool connect(std::string s){ return false; }
};

struct some_db2{
    bool connect(std::string ip, std::string user,
                 std::string pwd, std::string dbname){ return false; }
};

struct some_db3{
    bool connect(std::string ip, std::string user, std::string pwd,
                 std::string dbname, int timeout){ return false; }
};
```

Variadic Template


```
template<typename T>
struct proxy{
    template<typename... Args>
    decltype(auto) connect(Args&&... args){
        return t_.connect(std::forward<Args>(args)...);
    }

private:
    T t_;
};


proxy<some_db1> db1;
db1.connect("test.db");

proxy<some_db2> db2;
db2.connect("127.0.0.1", "test", "12345", "testdb");

proxy<some_db3> db3;
db3.connect("127.0.0.1", "test", "12345", "testdb", 5);
```

Structured Binding


```
std::map<int, int> mp = { {1, 2}, {3, 4} };
for(auto&& [k, v] : mp)
   std::cout<<k<<" "<<v<<'\n';


std::tuple<int, double> tp(1, 2.5);
auto [a, b] = tp;
```

Variadic Template + Structured Binding

- 分割一个定长的variadic template

```
template<typename... Args>
auto split_3args(Args&&... args){
    auto tp = std::make_tuple(std::forward<Args>(args)...);

    return std::make_tuple(std::get<0>(tp), std::get<1>(tp), std::get<2>(tp));
}


auto t = split_3args(1, 2.5, "test", '1'); //tuple(1, 2.5, "test")
```

Variadic Template + Structured Binding


```
template<typename... Args>
auto split_3args(Args&&... args){
    auto tp = std::make_tuple(std::forward<Args>(args)...);
    auto [a, b, c, d] = tp;
    return std::make_tuple(a, b, c);
}

auto split_3args(auto a, auto b, auto c, auto d){
    return std::make_tuple(a, b, c);
}
```

Constexpr If


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

template<typename T>
std::enable_if_t<!std::is_integral<T>::value, std::string> to_str(T t)
{
    return t;
}
```

Constexpr If


```
template<typename T>
auto to_str17(T t)
{
    if constexpr(std::is_integral<T>::value)
        return std::to_string(t);
    else
        return t;
}
```

- 消除enable_if
- 让编译期选择变得简单
- 更紧密、清晰的上下文

Enable_if

- 用constexpr if消除enable_if
- 提供参数相同，返回类型不同的同名接口

```
template<typename T, typename... Args>
constexpr std::enable_if_t<std::is_class_v<T>, std::vector<T>> query(Args&&... args){
    return {{std::forward<Args>(args)...}};
}

template<typename T, typename... Args>
constexpr std::enable_if_t<!std::is_class_v<T>, std::vector<T>> query(Args&&... args){
    return {std::forward<Args>(args)...};
}

auto v = query<int>(1,2,3); //vector<int>
auto v1 = query<A>(1);      //vector<A>
```

Constexpr If + Variadic Template

- 扩展接口：
  - 接口不变
  - 根据参数类型的不同展现不同的行为

```
extend();     //no extention
extend(0);    //do int branch
extend(2.5); //do double branch
```

Constexpr If + Variadic Template


```
template<typename... Args>
auto extend(Args&&... args){
    if constexpr(sizeof...(Args)==0){
        std::cout<<"no extention"<<std::endl;
    }
    else if constexpr(sizeof...(Args)==1){
        if constexpr(std::is_same_v<int, Args...>){
            std::cout<<"do int branch"<<std::endl;
        }
        else if constexpr(std::is_same_v<double, Args...>){
            std::cout<<"do double branch"<<std::endl;
        }
    }
}
```

Fold Expression

- 简化variadic template的使用

```
template<typename T>
auto add(T t){
    return t;
}

template<typename Arg, typename... Args>
auto add(Arg arg, Args... args){
    return arg+add(args...);
}

add(1,2,3); //6
```

```
template<typename... Args>
auto add(Args... args) {
    return (args +  ...);
}
```

```
template<typename... Args>
auto sub (Args... args) {
    return (args - ... - 1);
}
```

Fold Expression


```
//c++11
template<typename... Args>
void print(Args... Args){
   std::initializer_list<int>{(std::cout << args<<std::endl, 0)...};
}

//c++17 unary fold
template<typename... Args>
void print(Args... args) {
   ((std::cout<<args<<std::endl), ...);
}

//c++17 binary fold
template<typename ...Args>
void print(Args&&... args) {
   (std::cout << ... << args) << '\n';
}
```

- 连接任意个字符串

```
template<typename... Args>
inline void append(std::string& s, Args&&... args) {
    ((s+=std::forward<Args>(args), s += ", "),...);
}

std::string s="";
append(s, "a", "b", "c"); //a, b, c
```

Fold Expression

- 遍历tuple/variadic template

```
//c++14
template <typename... Args, typename Func, std::size_t... Idx>
void for_each(const std::tuple& t, Func&& f, std::index_sequence<Idx...>) {
    (void)std::initializer_list<int> { (f(std::get<Idx>(t)), void(), 0)...};
}

//c++17
template <typename... Args, typename Func, std::size_t... Idx>
void for_each(const std::tuple<Args...>& t, Func&& f, std::index_sequence<Idx...>){
    (f(std::get<Idx>(t)), ...);
}
```

Static_assert

- 编译期检查，在编译期就抓住错误

```
fun_for_1024<0,1,2,4>();  //ok
fun_for_1024<4,2,1,0>();  //ok

fun_for_1024<0,1,1,4>();  //static assertion failed: hi guy, just for 1024

fun_for_1024<1,0,2,5>();  //static assertion failed: hi guy, just for 1024

fun_for_1024<1,0,2>();    //static assertion failed: hi guy, just 4 numbers
```

Static_assert


```
template <size_t I, typename Seq> struct has_element;
template <size_t I, size_t... Is>
struct has_element<I, std::index_sequence <Is...>> :
        std::disjunction<std::is_same<std::integral_constant<size_t, I> ,
                std::integral_constant<size_t, Is>>...> {};

template<size_t ... Args>
constexpr auto fun_for_1024(){
    static_assert(sizeof...(Args)==4, "hi guy, just 4 numbers");

    static_assert(has_element<1, std::index_sequence <Args...>>::value&&
                  has_element<0, std::index_sequence <Args...>>::value&&
                  has_element<2, std::index_sequence <Args...>>::value&&
                  has_element<4, std::index_sequence <Args...>>::value,
                  "hi guy, just for 1024");
}
```

Future C++

- Type-and resource-safe
- Significantly simpler and clearer code
- As fast or faster than anything else
- Good at using “modern hardware”
- Significantly faster compilation catching many more errors

ORM

- 统一好用的接口
- 可扩展的接口
- 自动生成sql脚本
- 自动化地实体映射

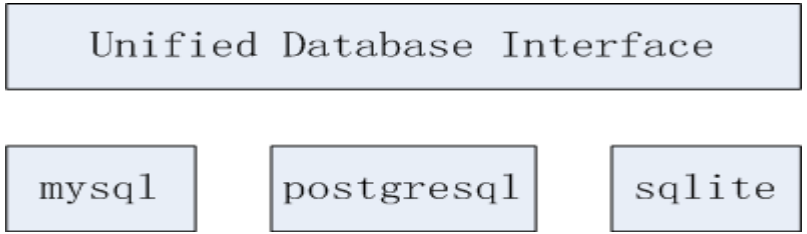
ORM


```
person p = {1, "test1", 2};
person p1 = {2, "test2", 3};
person p2 = {3, "test3", 4};
std::vector<person> v{p1, p2};

dbng<mysql> mysql;
mysql.connect("127.0.0.1", "dbuser", "yourpwd", "testdb");
mysql.create_datatable<person>();

mysql.insert(p);
mysql.insert(v);

mysql.update(p);
mysql.update(v);

auto result = mysql.query<person>(); //vector<person>
for(auto& person : result){
        std::cout<<person.id<<" "<<person.name<<" "<<person.age<<std::endl;
}

mysql.delete_records<person>();
```

https://github.com/qicosmos/ormpp

ORM

- 屏蔽不同数据库的差异




```
if (mysql_real_connect(con, "127.0.0.1", "root", "12345",
                       "testdb", 0, NULL, 0) == NULL)    {
    finish_with_error(con);
}
```

```
PGconn *conn = PQconnectdb("host=127.0.0.1 user=root password=12345 dbname=testdb");
if (PQstatus(conn) != CONNECTION_OK)
    error(PQerrorMessage(conn));
```

```
sqlite3* handle_ = NULL;
sqlite3_open("test.db", &handle_);
```

统一接口

- 通过可变模板参数统一接口
- 通过policy-base设计和variadic template来屏蔽数据库接口差异

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


dbng<mysql> mysql;
dbng<sqlite> sqlite;
dbng<postgresql> postgres;

TEST_REQUIRE(mysql.connect("127.0.0.1", "root", "12345", "testdb"));
TEST_REQUIRE(postgres.connect("127.0.0.1", "root", "12345", "testdb"));
TEST_REQUIRE(sqlite.connect("test.db"));
```

统一接口

- 通过constexpr if 和variadic template实现静态多态

```
template<typename... Args>
auto get_tp(int& timeout, Args&&... args) {
    auto tp = std::make_tuple(con_, std::forward<Args>(args)...);
    if constexpr (sizeof...(Args) == 5) {
        auto[c, s1, s2, s3, s4, i] = tp;
        timeout = i;
        return std::make_tuple(c, s1, s2, s3, s4);
    }
    else
        return tp;
}
```

扩展接口

- 通过constexpr if和variadic template来扩展接口

```
ormpp_key key{"id"};
ormpp_not_null not_null{{"id", "age"}};
ormpp_auto_increment_key auto_key{"id"};

mysql.create_datatable<person>();
mysql.create_datatable<person>(key);
postgres.create_datatable<person>(not_null);
sqlite.create_datatable<person>(not_null, key);
```

```
template<typename T, typename... Args>
constexpr auto create_datatable(Args&&... args){
    return db_.template create_datatable<T>(std::forward<Args>(args)...);
}
```

```
for_each(tp, [](auto item) {
    if constexpr (std::is_same_v<decltype(item), ormpp_not_null>) {
        //......
    }
    else if constexpr (std::is_same_v<decltype(item), ormpp_key>) {
        //......
    }
});
```

编译期反射

- 编译期获得对象的元数据（field name, field type, field sequence）

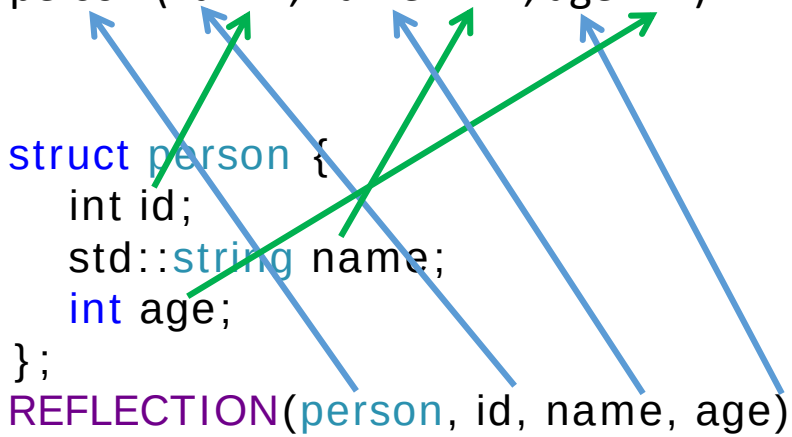
```
struct person {
   std::string name;
   int age;
};
REFLECTION(person, name, age)
std::cout << get_name<person, 0>()<< get_name<person, 1>() << std::endl;
std::cout << get<0>(p) << get<1>(p) << std::endl;

person p = { "admin", 20 };
for_each(p, [](const auto& item, auto idx) {
   std::cout << item << " " << decltype(i)::value << std::endl;
});
```

<<Compile-time Reflection, Serialization and ORM Examples>>
https://github.com/CppCon/CppCon2017
https://www.youtube.com/watch?v=WlhoWjrR41A

自动生成sql语句

- 通过编译期反射获取对象和字段的名称
- 通过类型映射获取数据库类型名称

" CREATE TABLE person (id INT, name TEXT, age INT) "



自动生成sql脚本

- 编译期反射获取对象名称

constexpr auto name = iguana::get_name<T>(); → "person"

- 编译期反射获取字段名数组

constexpr auto arr = iguana::get_array<T>(); → {"id" , "name" , "age" }

- 编译期获取类型名称数组？ →{“INTEGER" , “TEXT" , “INTEGER" }

类型映射

- C++类型映射到数据库字段类型

```
template <class T> struct identity{};

namespace ormpp_mysql {
    constexpr auto type_to_name(identity<char>) noexcept { return "TINYINT"sv; }
    constexpr auto type_to_name(identity<short>) noexcept { return "SMALLINT"sv; }
    constexpr auto type_to_name(identity<int>) noexcept { return "INTEGER"sv; }
    constexpr auto type_to_name(identity<float>) noexcept { return "FLOAT"sv; }
    constexpr auto type_to_name(identity<double>) noexcept { return "DOUBLE"sv; }
    constexpr auto type_to_name(identity<int64_t>) noexcept { return "BIGINT"sv; }
    constexpr auto type_to_name(identity<std::string>) noexcept { return "TEXT"sv; }
}

namespace ormpp_sqlite {
    constexpr auto type_to_name(identity<char>) noexcept { return "INTEGER"sv; }
    constexpr auto type_to_name(identity<short>) noexcept { return "INTEGER"sv; }
    constexpr auto type_to_name(identity<int>) noexcept { return "INTEGER"sv; }
    constexpr auto type_to_name(identity<float>) noexcept { return "FLOAT"sv; }
    constexpr auto type_to_name(identity<double>) noexcept { return "DOUBLE"sv; }
    constexpr auto type_to_name(identity<int64_t>) noexcept { return "INTEGER"sv; }
    constexpr auto type_to_name(identity<std::string>) noexcept { return "TEXT"sv; }
}

namespace ormpp_postgresql {
    constexpr auto type_to_name(identity<char>) noexcept { return "char"sv; }
    constexpr auto type_to_name(identity<short>) noexcept { return "smallint"sv; }
    constexpr auto type_to_name(identity<int>) noexcept { return "integer"sv; }
    constexpr auto type_to_name(identity<float>) noexcept { return "real"sv; }
    constexpr auto type_to_name(identity<double>) noexcept { return "double precision"sv; }
    constexpr auto type_to_name(identity<int64_t>) noexcept { return "bigint"sv; }
    constexpr auto type_to_name(identity<std::string>) noexcept { return "text"sv; }
}
```

获取类型名称

- 编译期获取类型名称数组

```
template <typename T>
inline constexpr auto get_type_names(DBType type){
    constexpr auto SIZE = iguana::get_value<T>();
    std::array<std::string_view, SIZE> arr = {};
    iguana::for_each(T{}, [&](auto& item, auto i){
        constexpr auto Idx = decltype(i)::value;
        using U = std::remove_reference_t<decltype(iguana::get<Idx>(std::declval<T>()))>;
        std::string_view s;
        switch (type){
            case DBType::mysql : s = ormpp_mysql::type_to_name(identity<U>{});
                break;
            case DBType::sqlite : s = ormpp_sqlite::type_to_name(identity<U>{});
                break;
            case DBType::postgresql : s = ormpp_postgresql::type_to_name(identity<U>{});
                break;
        }

        arr[Idx] = s;
    });

    return arr;
}
```

自动生成脚本

- 对象名 + 字段名 + 类型名 ➔ 自动生成sql语句

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

" CREATE TABLE person ( id INT, name TEXT, age INT) "
```

```
template<typename T, typename... Args>
constexpr auto create_datatable(Args&&... args){
    return db_.template create_datatable<T>(std::forward<Args>(args)...);
}
```

只需要一个对象类型就可以自动生成创建语句, insert,update,delete 类似

This is the magic of modern C++ !

实体映射


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

std::vector<person> v = mysql.query<person>();
std::vector<person> v1 = sqlite.query<person>();
std::vector<person> v2 = postgres.query<person>();

mysql.query<person>("id=1", "limit 10");
sqlite.query<person>("id>1", "limit 10");
postgres.query<person>("id=2", "limit 10");
```

```
template<typename T, typename... Args>
constexpr auto query(Args&&... args){
    return db_.template query<T>(std::forward<Args>(args)...);
}
```

实体映射

- Postgresql to Entity

```
std::vector<T> v;
auto ntuples = PQntuples(res_);

for(auto i = 0; i < ntuples; i++){
    T t = {};
    iguana::for_each(t, [this, i, &t](auto item, auto I)
    {
        assign(t.*item, i, decltype(I)::value);
    });
    v.push_back(std::move(t));
}
```

实体映射

- Postgresql to Entity

```
template<typename T>
constexpr void assign(T&& value, int row, size_t i){
    using U = std::remove_const_t<std::remove_reference_t<T>>;
    if constexpr(std::is_integral_v<U>&&!is_int64_v<U>){
        value = std::atoi(PQgetvalue(res_, row, i));
    }
    else if constexpr (is_int64_v<U>){
        value = std::atoll(PQgetvalue(res_, row, i));
    }
    else if constexpr (std::is_floating_point_v<U>){
        value = std::atof(PQgetvalue(res_, row, i));
    }
    else if constexpr(std::is_same_v<std::string, U>){
        value = PQgetvalue(res_, row, i);
    }
    else {
        std::cout<<"this type has not supported yet"<<std::endl;
    }
}
```

实体映射

- Sqlite to Entity

```
std::vector<T> v;
while (true)
{
    result = sqlite3_step(stmt_);
    if (result == SQLITE_DONE)
        break;

    if (result != SQLITE_ROW)
        break;

    T t = {};
    iguana::for_each(t, [this, &t](auto item, auto I)
    {
        assign(t.*item, decltype(I)::value);
    });

    v.push_back(std::move(t));
}
```

实体映射

- Sqlite to Entity

```
template<typename T>
constexpr void assign(T&& value, size_t i){
    using U = std::remove_const_t<std::remove_reference_t<T>>;
    if constexpr(std::is_integral_v<U>&&!is_int64_v<U>){
        value = sqlite3_column_int(stmt_, i);
    }
    else if constexpr (is_int64_v<U>){
        value = sqlite3_column_int64(stmt_, i);
    }
    else if constexpr (std::is_floating_point_v<U>){
        value = sqlite3_column_double(stmt_, i);
    }
    else if constexpr(std::is_same_v<std::string, U>){
        value.reserve(sqlite3_column_bytes(stmt_, i));
        value.assign((const char*)sqlite3_column_text(stmt_, i), (size_t)sqlite3_column_bytes(stmt_, i));
    }
    else {
        std::cout<<"this type has not supported yet"<<std::endl;
    }
}
```

实体映射

- 多表查询

```
//std::vector<std::tuple<person, student>>
auto result = mysql.query<std::tuple<person, student>>("select * from person, student"s);

auto sql = "select person.*, student.name, student.age from person, student";
auto result1 = mysql.query<std::tuple<person, std::string, int>>(sql);
```

```
template<typename T, typename... Args>
constexpr std::enable_if_t<iguana::is_reflection_v<T>, std::vector<T>>
query(Args&&... args){...}

template<typename T, typename Arg, typename... Args>
constexpr std::enable_if_t<!iguana::is_reflection_v<T>, std::vector<T>>
query(const Arg& s, Args&&... args){...}
```

实体映射

- 返回tuple结果时要注意元素是否为对象

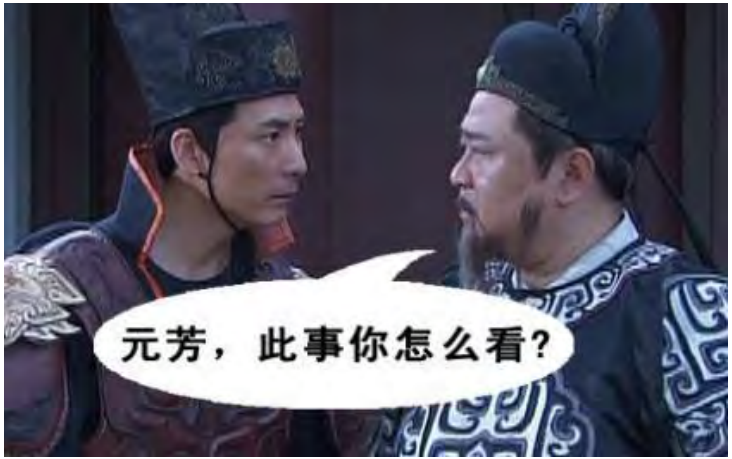
```
T tp = {};
size_t index = 0;
iguana::for_each(tp, [this, &index](auto& item, auto I)
{
    if constexpr(iguana::is_reflection_v<decltype(item)>){
        std::remove_reference_t<decltype(item)> t = {};
        iguana::for_each(t, [this, &index, &t](auto ele, auto i)
        {
            assign(t.*ele, index++);
        });
        item = std::move(t);
    }else{
        assign(item, index++);
    }
}, std::make_index_sequence<SIZE>{});

v.push_back(std::move(tp));
```

ormpp

- ormpp ---- 让数据库操作变得简单
  - header only
  - cross platform
  - unified interface
  - easy to use
  - easy to change database

https://github.com/qicosmos/ormpp

- Logging
- Validation
- Thread Strategy
- Caching
- Exception Handling
- … and a lot more


元芳，此事你怎么看?

```
dbng<mysql> mysql;
mysql.connect("127.0.0.1", "root", "12345", "testdb");
```

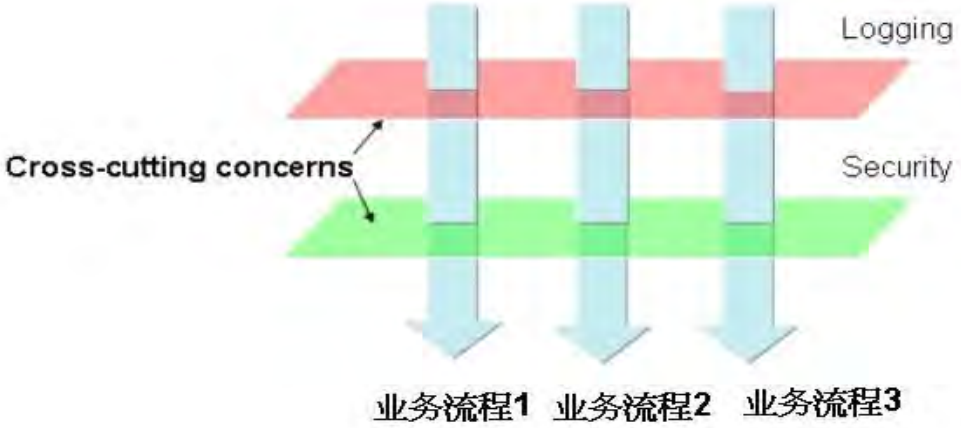
```
mysql.warper_connect<log>("127.0.0.1", "root", "12345", "testdb");
```

```
mysql.warper_connect<validate, log>("127.0.0.1", "root", "12345",
"testdb");
```

```
validate before
log before
log after
validate after
```

```
mysql.warper_connect<validate, log, thead_proxy>("127.0.0.1", "root", "12345",
"testdb");
```

AOP


Logging
Cross-cutting concerns
Security
业务流程1
业务流程2
业务流程3

AOP


```
struct log{
    template<typename... Args>
    bool before(Args... args){
        std::cout<<"log before"<<std::endl;
        return true;
    }

    template<typename T, typename... Args>
    bool after(T t, Args... args){
        std::cout<<"log after"<<std::endl;
        return true;
    }
};

struct validate{
    template<typename... Args>
    bool before(Args... args){
        std::cout<<"validate before"<<std::endl;
        return true;
    }

    template<typename T, typename... Args>
    bool after(T t, Args... args){
        std::cout<<"validate after"<<std::endl;
        return true;
    }
};
```

before和after你可以定义一个也可以定义两个

AOP

- 核心逻辑之前的切面

```
template<typename... AP, typename... Args>
auto warper(Args&&... args){
    using result_type = decltype(std::declval<decltype(this)>()->
            connect(std::declval<Args>()...));

    //before
    bool r = true;
    std::tuple<AP...> tp{};
    for_each(tp, [&r, &args...](auto& item){
        if(!r)
            return;

        if constexpr (has_before<decltype(item)>::value)
        r = item.before(std::forward<Args>(args)...);
    }, std::make_index_sequence<sizeof...(AP)>{});

    if(!r)
        return result_type{};
```

AOP

- 核心逻辑之后的切面

```
//business
auto lambda = [this, &args...]{ return this->connect(std::forward<Args>(args)...); };
result_type result = std::invoke(lambda);

//after
for_each_r(tp, [&r, &result, &args...](auto& item){
    if(!r)
        return;

    if constexpr (has_after<decltype(item), result_type>::value)
    r = item.after(result, std::forward<Args>(args)...);
}, std::make_index_sequence<sizeof...(AP)>{});
```

```
#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)
```

Purecpp----Modern C++ Open Source Community

- purecpp open source
  - Rpc framework—rest_rpc
  - Serialization engine—iguana
  - ORM--ormpp
  - http framework--cinatra

http://purecpp.org (modern c++ open source community)
https://github.com/qicosmos

https://www.youtube.com/watch?v=WlhoWjrR41A
https://www.youtube.com/watch?v=vh1BhlqF-fs

https://isocpp.org/