HOLO

发布 *1*

Darwin Yuan

2020 年 09 月 17 日

Contents

注解： holo 是一个编译时类型操控库。

C++ 编译时元编程极为强大：即便对于类型计算能力也是图灵完备的。而对于类型，尤其是类型列表的强大操作能力，有助于实现各种无论从空间效率，还是时间效率都极好的各种 *DSL* 。

但同时，这又是一种专家级技能。模版元编程技术让很多新手望而生畏。虽然其计算思想和表达能力与函数式编程完全一致，但其并不简洁直观的语法，即便让精通此项技术的 *C++* 程序员也希望其能得到改善。

另外，编译时元编程的一个最大缺点是其编译时间。进行复杂编译时计算的程序，会让编译时间变得完全不可忍受。尽管这会让运行时零成本开销，最终带来收益，但毕竟编译等待时间也是一个非常影响程序员效率和心情的因素。

因而，一个好的编译时元编程库，需要同时兼顾两个目标：

1. 让程序员可以以编写运行时程序的方式进行元编程：不（仅仅）是操作数据，而是操作类型；
2. 必须让编译时间处于一个可接受的水平（当然越快越好）；

`holo` 正是为这两个目标而生。

通过 `holo` ，你可以以如下方式操作类型：

```
constexpr auto result = holo::append( holo::type_c<long>,
                                      holo::type_list_t<int, double, float>);
static_assert(result == holo::type_list_t<int, double, float, long>);
```

以下是来自于现实项目 graph-dsl 的例子。你可以看出，`holo` 完全支持以 `ranges` 的 `pipeline` 语法对操作进行组合，在 **编译**时，对 **类型**进行各种计算和操作：

```
constexpr static auto sorted_non_leaf_nodes =
   all_decedents_map
   | holo::sort([](auto l, auto r) {
      return holo::contains(holo::first(l), holo::second(r)); })
   | holo::transform([](auto elem) {
      return holo::first(elem); })
   | holo::reverse();


constexpr static auto root_nodes =
   __HOLO_tuple_t<NODES...>
   | holo::filter([](auto elem){
      return decltype(elem)::type::is_root == holo::true_c; })
   | holo::transform([](auto elem){
      return holo::type_c<typename decltype(elem)::type::node_type>;
   });
```

(下页继续)

(续上页)

```
constexpr static auto sorted_tagged_intermediate_nodes =
   sorted_non_leaf_nodes
   | holo::remove_if([](auto elem) {
      return holo::contains(elem, root_nodes); })
   | holo::transform([](auto elem) {
      return holo::type_c<node_trait<typename decltype(elem)::type, node_
↪category::Intermediate>>;
   });


constexpr static auto leaf_tagged_nodes =
   all_decedents_map
   | holo::fold_left(__HOLO_make_tuple(), [](auto acc, auto elem){
      return holo::concat(acc, holo::second(elem)); })
   | holo::unique()
   | holo::remove_if([](auto elem) {
      return holo::contains(elem, sorted_non_leaf_nodes); })
   | holo::transform([](auto elem) {
      return holo::type_c<node_trait<typename decltype(elem)::type, node_
↪category::Leaf>>;
   });
```