TMP with Boost Mp11

VIKRAM KALLURU

MAYSTREET (NOW LSEG)

Agenda

Compare solutions that use conventional metaprogramming and Boost Mp11

Problem Statement

Given two lists of IP Addresses

- find the intersection of the IPs and
- obtain the index in List 1

List 1: [224.0.0.1, 192.168.0.2, 10.100.100.1, 10.100.100.2]

List 2: [192.168.0.3, 224.0.0.1, 10.100.100.3, 10.100.100.1]

Intersected List: [224.0.0.1, 10.100.100.1]

Indices from List 1: [0, 2]

And print a sample config

[feed/nasdaq]

sessions/0/

sessions/2/

Data Representation

```
82  template <int, int, int, int>
83  struct IP{};
84
85  template <typename… Ts>
86  struct TypeList {};
87
88  using List1 = Typelist< IP<192, 168, 1, 1>,
89                          IP<10, 0, 0, 2>,
90                          IP<224, 0, 0, 1>,
91                          IP<172, 16, 0, 1> >;
92
93  using List2 = Typelist< IP<10, 100, 100, 1>,
94                          IP<192, 168, 1, 1>,
95                          IP<172, 16, 0, 1> >;
```

Conventional Solution

```
63      using List1 = Typelist<IP<192, 168, 1, 1>,
64                             IP<10, 0, 0, 2>,
65                             IP<224, 0, 0, 1>,
66                             IP<172, 16, 0, 1>>;
67
68      using List2 = Typelist<IP<10, 100, 100, 1>,
69                             IP<192, 168, 1, 1>,
70                             IP<172, 16, 0, 1>>;
71
72      using Intersection = Intersect<List1, List2>;
73      static_assert(std::is_same_v< Intersection, Typelist< IP<172, 16, 0, 1> , IP<192, 168, 1, 1> > > );
```

Conventional Solution

```
31  template<typename List1, typename List2>
32  struct Filter;
33
34  template<typename List2>
35  struct Filter<Typelist<>, List2> {
36      using type = Typelist<>;
37  };
38
39  template<typename Head, typename... Tail, typename List2>
40  struct Filter<Typelist<Head, Tail...>, List2> {
41      using type = std::conditional_t<
42          Contains<Head, List2>::value,
43          typename Concat<typename Filter<Typelist<Tail...>, List2>::type, Head>::type,
44          typename Filter<Typelist<Tail...>, List2>::type
45      >;
46  };
47
48  template<typename List1, typename List2>
49  using Intersect =  Filter<List1, List2>::type;
```

Conventional Solution

```
18  template<typename T, typename List>
19  struct Contains;
20
21  template<typename T>
22  struct Contains<T, Typelist<>> : std::false_type {};
23
24  template<typename T, typename Head, typename... Tail>
25  struct Contains<T, Typelist<Head, Tail...>> : std::conditional_t<
26      std::is_same_v<T, Head>,
27      std::true_type,
28      Contains<T, Typelist<Tail...>>
29  > {};
30
31  template<typename Head, typename... Tail, typename List2>
32  struct Filter<Typelist<Head, Tail...>, List2> {
33      using type = std::conditional_t<
34          Contains<Head, List2>::value,
35          typename Concat<typename Filter<Typelist<Tail...>, List2>::type, Head>::type,
36          typename Filter<Typelist<Tail...>, List2>::type
37      >;
38  };
```

Conventional Solution

```
31  template<typename List, typename T>
32  struct Concat;
33
34  template<typename... Ts, typename T>
35  struct Concat<Typelist<Ts...>, T> {
36      using type = Typelist<Ts..., T>;
37  };
38
39  template<typename Head, typename... Tail, typename List2>
40  struct Filter<Typelist<Head, Tail...>, List2> {
41      using type = std::conditional_t<
42          Contains<Head, List2>::value,
43          typename Concat<typename Filter<Typelist<Tail...>, List2>::type, Head>::type,
44          typename Filter<Typelist<Tail...>, List2>::type
45      >;
46  };
47
```

Mp11 solution

```
7   template <int A, int B, int C, int D>
8   struct IP {};
9
10  using List1 = mp_list<
11      IP<192, 168, 1, 1>,
12      IP<10, 0, 0, 1>,
13      IP<172, 16, 0, 1>
14  >;
15
16  using List2 = mp_list<
17      IP<192, 168, 1, 1>,
18      IP<10, 0, 0, 2>,
19      IP<172, 16, 0, 1>
20  >;
21
22  template<typename T>
23  using is_in_list_2 = mp_contains<List2, T>;
24
25  using Intersection = mp_copy_if<List1, is_in_list_2>;
26
27  static_assert(std::is_same_v<Intersection, mp_list< IP<192, 168, 1, 1>, IP<172, 16, 0, 1> > >);
```

Mp11 solution

```
7  template <int A, int B, int C, int D>
8  struct IP {};
9
10 using List2 = mp_list<
11     IP<192, 168, 1, 1>,
12     IP<10, 0, 0, 2>,
13     IP<172, 16, 0, 1>
14 >;
15
16 using List1 = mp_list<
17     IP<192, 168, 1, 1>,
18     IP<10, 0, 0, 1>,
19     IP<172, 16, 0, 1>
20 >;
21
22 using Intersection = mp_set_intersection<List1, List2>;
23 static_assert(std::is_same_v<Intersection, mp_list< IP<192, 168, 1, 1>, IP<172, 16, 0, 1> > >);
24
```

Problem Statement

Given two lists of IP Addresses

- find the intersection of the IPs and
- obtain the index in List 1

List 1: [224.0.0.1, 192.168.0.2, 10.100.100.1, 10.100.100.2]

List 2: [192.168.0.3, 224.0.0.1, 10.100.100.3, 10.100.100.1]

[feed/nasdaq]

sessions/0/

sessions/2/

Mp11 solution

```
22  using SetIntersection = mp_set_intersection<List1, List2>;
23  static_assert(std::is_same_v<SetIntersection,
24                               mp_list<IP<192, 168, 1, 1>, IP<172, 16, 0, 1> > >);
25
26  int main() {
27      std::cout << "[feed/nasdaq]" << std::endl;
28      boost::mp11::mp_for_each<SetIntersection>([](auto I) {
29      std::cout << "sessions/";
30      std::cout << mp_find<List1, decltype(I)>::value << std::endl;
31      });
32  }
33
```

Links

Conventional Solution

MP11 solution

Boost MP11 Reference