20220605-元编程

1.学习内容

1.1 元编程

顺序、分支与循环代码

C++ | 复制代码

```cpp
/*顺序执行的代码*/
template <typename T>
struct RemoveReferenceConst_
{
private:
    using inter_type=typename std::remove_reference<T>::type
public:
    using type = typename std::remove_const<inter_type>::type;
    //注意这里两行代码不能调换顺序。在编译期，编译器会扫描两遍结构体中的代码，第一遍处理声明
    //第二遍才会深入到函数的定义之中。在下面的runtime例子中，第一遍时，编译器只是了解到包含了两个成员函数
    //在后续的扫描中，才会关注fun1中调用了fun2。正是因为这样的扫描，编译器不会报告错误。
    //而如果把上面的代码调换顺序，编译器首次从前到后扫描程序时，会发现type依赖于没有定义的inter_type
    //因此不继续扫描而是直接报错
};

template <typename T>
using RemoveReferenceConst = typename RemoveReferenceConst_<T>::type;

RemoveReferenceConst<const int&> h = 3;

struct RunTimeExample
{
    static void fun1() { fun2(); }
    static void fun2() { std::cerr << "hello" << std::endl; }
};
```

C++ | 复制代码

```cpp
/*分支执行的代码*/
//使用std::conditional来实现分支
namespace test
{
    //如果B为false，函数返回F，否则返回T
    template <bool B,typename T,typename F>
    struct conditional
    {
        using type = T;
    };
    template <typename T,typename F>
    struct conditional <false, T, F>
    {
        using type = F;
    };
    template <bool B,typename T,typename F>
    using conditional_t = typename conditional<B, T, F>::type;
}
test::conditional<true, int, float>::type x = 4;
test::conditional<false, int, float>::type y = 1.0f;
test::conditional_t<false, int, float> z = 1.0f;

//使用特化实现分支
struct A; struct B;
template<typename T>
struct Fun_
{
    constexpr static size_t value = 0;
};
template<>
struct Fun_<A>
{
    constexpr static size_t value = 1;
};
template<>
struct Fun_<B>
{
    constexpr static size_t value = 2;
};

constexpr size_t h = Fun_<B>::value;
————————另一种方式————————————————
struct A; struct B;
template<typename T>
constexpr size_t Fun = 0;
```

```
46
47    template<>
48    constexpr size_t Fun<A>= 1;
49
50    template<>
51    constexpr size_t Fun<B>= 2;
52
53    constexpr size_t h = Fun<B>;
```

2.结果描述

最近一段学习效率颇为低下，原本应该充满热情，结果却沦为对自己的一种应付。要好好反思一下。