动手实现is_same、remove_const等type_traits

原创 高效Cpp 高效Cpp 2022-10-08 06:00 发表于安徽

收录于合集
#C++入门到进阶 13 #C++STL精讲系列 6

往往简单的代码能发挥很大的作用，具体讲解，可参考[阅读原文]视频。

```cpp
#include <iostream>
#include <type_traits>


namespace cpp {
    // 主模板
    template <class T, class U>
    struct is_same {
        static constexpr bool value = false;
    };
    // 特化
    template <class T>
    struct is_same<T, T> {
        static constexpr bool value = true;
    };
    // 别名
    template <class T, class U>
    constexpr bool is_same_v = is_same<T, U>::value;
    // remove const/ remove volatile / remove_reference

    template<class T>
    struct integral_const {
        using type = T;
    };

    template <class T>
    struct remove_reference : integral_const<T>{};

    template<class T>
    struct remove_reference<T&>: integral_const<T>{};


    template<class T>
    using remove_reference_t = remove_reference<T>::type;

    template <class T>
    struct remove_const : integral_const<T>{};

    template<class T>
    struct remove_const<const T>: integral_const<T>{};
}
int main()
{

    std::remove_const<std::remove_reference_t<const int &>>::type

    std::is_same<int, int&>;

    return 0;
}

// class A {
//     public:
//     A() = delete;
//     double add(int a, int b){
//         std::cout << __FUNCTION__;
//         return a + b;
//     }
// };
// int main()
// {
//     constexpr bool v = std::is_same_v<double,  std::decay_t<de
//     return 0;
// }
// template<typename T, typename U>
// auto add(T t, U u) {
//     std::cout << __FUNCTION__;
//     return t + u;
// }
// int main()
// {
//     constexpr bool v = std::is_same_v< decltype(add(1, 1)),  d
//     return 0;
// }
// int main()
// {
//     std::is_same_v<int, decltype(1+1.0)>;
//     return 0;
// }
// int main()
// {
//     //  int& -> int
//     std::is_same_v<int, std::decay_t<volatile int&>>;

//     constexpr bool v = typeid(int) == typeid(volatile int);
//     return 0;
// }
// template <typename T>
// bool is_zero(T v)
// {
//     if (std::is_same_v<T, float>) {
//         return (fabs(v) < FLT_EPSILON);
//     }
//     else if (std::is_same_v<T, double>) {
//         return (fabs(v) < DBL_EPSILON);
//     }
//     return v == 0;
// }
// int main()
// {
//     if(is_zero(0.000'0001f))
//     {
//         std::cout << "1" << std::endl;
//     }
//     if (is_zero(0)) {
//         std::cout << "2" << std::endl;
//     }
//     return 0;
// }
```

收录于合集 #C++入门到进阶 13

上一篇
宏定义污染与规避

下一篇
is_same 判断类型是否相同

阅读原文