第38章 类模板的模板实参推导

《现代C++语言核心特性解析》 谢丙堃

通过初始化构造推导类模板的模板实参

- C++17标准之前，实例化类模板必须显式指定模板实参：

```
std::tuple<int, double, const char*> v{5, 11.7, "hello world"};
```

- 变通的方法，使用函数模板：

```
auto v = std::make_tuple(5, 11.7, "hello world");
```

- C++17标准支持了类模板的模板实参推导

```
std::tuple v{ 5, 11.7, "hello world" };
```

通过初始化构造推导类模板的模板实参

- 实参推导对非类型形参的类模板同样适用

```
template<class T, std::size_t N>
struct MyCountOf
{
    MyCountOf(T(&)[N]) {}
    std::size_t value = N;
};

int main()
{
    MyCountOf c("hello");
    std::cout << c.value << std::endl;
}
```

通过初始化构造推导类模板的模板实参

- 类模板的模板实参不允许部分推导：

```
template<class T1, class T2>
struct foo
{
    foo(T1, T2) {}
};
foo<int> v3(5, 6.8);              // 编译错误
```

拷贝初始化优先

- 考虑以下代码v2的类型：
  - std::vector<int>
  - std::vector<std::vector<int>>

```
std::vector v1{ 1, 3, 5 };
std::vector v2{ v1 };
```

对lambda类型的用途

- 用于存储lambda表达式对象

```
template<class T>
struct LambdaWarp
{
    LambdaWarp(T t) : func(t) {}
    template<class ... Args>
    void operator() (Args&& ... arg)
    {
        func(std::forward<Args>(arg)...);
    }
    T func;
};
```

```
// 显式指定
auto l = [](int a, int b) {
        std::cout << a + b << std::endl;
    };
LambdaWarp<decltype(l)> x(l);

// 自动推导
LambdaWarp x([](int a, int b) {
        std::cout << a + b << std::endl;
    });
```

别名模板的类模板实参推导

- 结合了别名模板和类模板实参推导:

```
template <class T, class U> struct C {
    C(T, U) {}
};

template<class V>
using A = C<V*, V*>;

int i{};
double d{};
A a1(&i, &i);   // 编译成功，可以推导为A<int>
A a2(i, i);     // 编译失败，i无法推导为V*
A a3(&i, &d);   // 编译失败，(int *, double *)无法推导为(V*, V*)
```

聚合类型的类模板实参推导

- 根据初始化列表推导出模板实参

```
template <class T>
struct S {
    T x;
    T y;
};

S s1{ 1, 2 };   // 编译成功 S<int>
S s2{ 1, 2u };  // 编译失败
```

感谢您的观看

欢迎关注