C++ 学习笔记

keepone

2021 年 2 月 4 日

目录

1 STL

1.1 utility

1.1.1 variant

2 SFINAE

2.1 简介

考虑如下的代码：

如果声明一个 foo 的重载函数，foo(0) 就会编译通过。但是编译器在编译时，也只是看到了包含模板的头文件，并没有找到 foo 的重载，为什么没有报错。

2.2 函数重载机制

- 名称查找，将找到所有模板与非模板函数。
- 模板类型的推导，函数模板由实参推导而来。
  - 推导模板的所有参数，包括返回类型及参数类型
  - 当推导失败时，将当前模板从函数重载集中删除，但并不会报错
- 最终的函数重载集，这里会看到所有函数。
- 选择最合适的函数

4.1.2 引用空悬

- 向下（入栈）传递不会导致空悬。
- 向上（出栈）、多线程交叉传递可能导致空悬。

4.1.3 引用捕获与拷贝捕获

kitten 的捕获发生在生调用时，此时 g=20，cat 的捕获发生在声明时，此时 g=10。

4.1.4 局部静态变量

局部静态变量的行为相比于类的静态变量来理解

4.1.5 拷贝与移动

Lambda 是否可以拷贝、是否可移动取决于它的捕获对象是否可拷贝、是否可移动。

4.2 高阶使用

4.2.1 各版本特性

- c++14
  - generic lambdas: pass auto argument, compiler expands to a function template
  - capture with initialiser: with this fiture you can capture not only existing variables from the other scope, but also create new state variables for lambdas. This also allowed capturing moveable only types
- c++17
  - constexpr lambdas: lambdas can work in a constexpr context.
  - capturing this improvements: capture *this by copy, avoiding dangling when returning the lambda from a member function or store it
- c++20
  - template lambdas: improvements to generic lambdas which offers move control over the input template argument
  - lambdas and concepts: lambdas can also work with constrained auto and concepts, so they are as flexible as functors as template functions
  - lambdas in unevaluated contexts: can now create a map or a set and use a lambda as a predicate

4.2.2 捕获 **this**

Lambda 表达式中的 this 并不代表 lambda 表达式本身，而是代表定义它的那个定义域。下列 Widget 中的两个 this 都代表 Widget。

捕获 this 的多种方式

4.2.3 捕获变参

4.2.4 Lambda 作为函数实参

4.2.5 重载 Lambda

4.2.6 继承 Lambda

4.3 Lambda vs Closure(闭包)

闭包是编程的一个概念，来自于函数式编程，c++ 中的 Lambda 表达式就是一种闭包。

- c++ Lambda 是一个表达式，该表达式被编译后生成一个闭包以在运行时工作。
- lambda 与闭包类似于类与类的实现，前者只存在于代码中，不存在于运行时。
- 上式中的 f 并不是闭包，而是闭包的拷贝，闭包是一个临时量。如果要保存闭包，可以使用万能引用实现，如下图所示。

4.4 Lambda vs std::function

4.4.1 异同

- std::function 只能存储可拷贝对象，lambda 可存储任意对象
- std::function 永远可以拷贝，lambda 的拷贝与移动性取决于其捕获对象的拷贝、移动性

4.4.2 传递成员函数