C++中的 lambda 表达式

王　昊 17307130027

在 C++ 中 lambda 函数有点类似匿名函数。
我们无需定义一个函数结构随后再去使用这个函数，而是直接在使用时将其声明使用。
lambda 是字面上定义一个函数而不是持有一个函数，也就是我们可以使用函数指针的场景是可以使用 lambda 来代替的。

形式

[函数对象参数] (操作符重载函数参数) mutable 或 exception 声明 -> 返回值类型 {函数体}

- [函数对象参数]

  标识一个 Lambda 表达式的开始，这部分必须存在，不能省略。函数对象参数是传递给编译器自动生成的函数对象类的构造函数的。函数对象参数只能使用那些到定义 Lambda 为止时 Lambda 所在作用范围内可见的局部变量(包括 Lambda 所在类的 this)。

  函数对象参数有以下形式：

  - 空。没有任何函数对象参数。
  - =。函数体内可以使用 Lambda 所在范围内所有可见的局部变量（包括 Lambda 所在类的 this），并且是值传递方式（相
    当于编译器自动为我们按值传递了所有局部变量）。
  - &。函数体内可以使用 Lambda 所在范围内所有可见的局部变量（包括 Lambda 所在类的 this），并且是引用传递方式
    （相当于是编译器自动为我们按引用传递了所有局部变量）。
  - this。函数体内可以使用 Lambda 所在类中的成员变量。
  - a。将 a 按值进行传递。按值进行传递时，函数体内不能修改传递进来的 a 的拷贝，因为默认情况下函数是 const 的，要修改传递进来的拷贝，可以添加 mutable 修饰符。
  - &a。将 a 按引用进行传递。
  - a，&b。将 a 按值传递，b 按引用进行传递。
  - =，&a，&b。除 a 和 b 按引用进行传递外，其他参数都按值进行传递。
  - &，a，b。除 a 和 b 按值进行传递外，其他参数都按引用进行传递。

- (操作符重载函数参数)

  标识重载的 () 操作符的参数，没有参数时，这部分可以省略。参数可以通过按值（如: (a, b)）和按引用 (如: (&a, &b)) 两种方式进行传递。

- mutable 或 exception 声明

  这部分可以省略。按值传递函数对象参数时，加上 mutable 修饰符后，可以修改传递进来的拷贝（注意是能修改拷贝，而不是值本身）。exception 声明用于指定函数抛出的异常，如抛出整数类型的异常，可以使用 throw(int)。

- -> 返回值类型

  标识函数返回值的类型，当返回值为 void，或者函数体中只有一处 return 的地方（此时编译器可以自动推断出返回值类型）时，这部分可以省略。

- {函数体}

  标识函数的实现，这部分不能省略，但函数体可以为空。

示例

```
[](int x,int y){return x+y;}//隐式返回类型
[](int x,int y)->int{int z=x+y;return z;}//上方变种
[](int& x){++x;}//没有 return 语句，返回值为 void
[](){++global_x;}
[]{++global_x;}//两种相同表达，仅访问某个全局变量
[&,value,this](int x){total+=x*value*this->some_func();}
```

捕获列表

- 值捕获（类似值传递）

  值捕获和参数传递中的值传递类似，被捕获的变量的值在 Lambda 表达式创建时通过值拷贝的方式传入，因此随后对该变量的修改不会影响影响 Lambda 表达式中的值。

```
int testFunc1()
{
    int nTest1 = 23;
    auto f = [nTest1] (int a, int b) -> int
    {
        return a + b + 42 + nTest1;
        //nTest1 = 333;                    不能在 lambda 表达式中修改捕获变量的值
    };
    cout << f(4, 3) << "&nTest1=" << nTest1 << endl;
}
```

  需要注意的是，不能在 lambda 表达式中修改捕获变量的值。lambda 表达式中的代码是在一个单独的函数中执行的，这个函数在调用时创建了自己的栈帧，而其使用的 nTest1 局部变量在 testFunc1 的栈帧中，捕获列表中出现的局部变量一定会通过某种方式传递给 lambda 匿名类。nTest 是在 lambda 匿名类构造时传入的。并且传入的是 nTest1 的引用 lambda 匿名类会将捕获参数中的变量添加到其成员变量中，并设置一个带有该参数引用类型的构造函数。

  值捕获时，C++编译器在构建 lambda 表达式的匿名类时将局部变量的引用传入，并在构造函数中完成对相应成员变量的赋值。在调用其 operator()函数时，如果用到了捕获列表中的局部变量，则从给匿名类对象的成员变量中取出。

- 引用捕获（类似引用）

  使用引用捕获一个外部变量，需在捕获列表变量前面加上一个引用说明符&。

```
void fnTest()
{
    int nTest1 = 23;
    auto f = [&nTest1] (int a, int b) -> int
    {
        cout << "In functor before change nTest=" << nTest1 << endl; //nTest1=23333
        nTest1 = 131;
        cout << "In functor after change nTest=" << nTest1 << endl; // nTest1 = 131
        return a + b + 42 + nTest1;
    };
    nTest1 = 23333;
    cout << f(4, 3) << "&nTest1=" << nTest1 << endl;          //nTest1 = 23333
```

}

lambda 表达式匿名对象在构造函数中依然传入了 nTest 的地址（引用）

由于构造函数调用时，传入的是 nTest1 的地址，这里渠道地址后，直接存到了成员对象中，是通过成员变量中的指针完成的。

- 外部捕获

上面的值捕获和引用捕获都需要我们在捕获列表中显示列出 Lambda 表达式中使用的外部变量。除此之外，我们还可以让编译器根据函数体中的代码来推断需要捕获哪些变量，这种方式称之为隐式捕获。隐式捕获有两种方式，分别是[=]和[&]。[=]表示以值捕获的方式捕获外部变量，[&]表示以引用捕获的方式捕获外部变量。

一个无指定任何捕获的 lambda 函数可以显示转换成一个具有相同形式的函数指针。

e.g.

```
auto a_lambda_func = [](int x) { /* ... */ };
void (*func_ptr)(int) = a_lambda_func;
func_ptr(4);
```

例子是合法的。

Reference：

Support For C++11/14/17 Features (Modern C++)
http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2012/n3337.pdf
https://zh.wikipedia.org/wiki/%CE%9B%E6%BC%94%E7%AE%97
https://www.jianshu.com/p/6482fbd3abdf