

10 | lambda：函数式编程带来了什么？

2020-05-28 罗剑锋

罗剑锋的C++实战笔记

[进入课程 >](#)



讲述：Chrono

时长 13:06 大小 12.00M



你好，我是 Chrono。

在 [第 1 节课](#) 的时候，我就说到过“函数式编程”，但只是简单提了提，没有展开讲。

作为现代 C++ 里的五种基本编程范式之一，“函数式编程”的作用和地位正在不断上升，而且在其他语言里也非常流行，很有必要再深入研究一下。

掌握了函数式编程，你就又多了一件“趁手的兵器”，可以更好地运用标准库里的容器和算法，写出更灵活、紧凑、优雅的代码。



所以，今天我就和你聊聊函数式编程，看看它给 C++ 带来了什么。

C++ 函数的特殊性


说到“函数式编程”，那肯定就要先从函数（function）说起。

C++ 里的函数概念来源于 C，是面向过程编程范式的基本部件。但严格来说，它其实应该叫“子过程”（sub-procedure）、“子例程”（sub-routine），是命令的集合、操作步骤的抽象。

函数的目的是封装执行的细节，简化程序的复杂度，但因为它有入口参数，有返回值，形式上和数学里的函数很像，所以就被称为“函数”。

在语法层面上，C/C++ 里的函数是比较特别的。虽然有函数类型，但不存在对应类型的变量，不能直接操作，只能用指针去间接操作（即函数指针），这让函数在类型体系里显得有点“格格不入”。

函数在用法上也有一些特殊之处。在 C/C++ 里，所有的函数都是全局的，没有生存周期的概念（static、名字空间的作用很弱，只是简单限制了应用范围，避免名字冲突）。而且函数也都是平级的，不能在函数里再定义函数，也就是**不允许定义嵌套函数、函数套函数**。


 复制代码

```
1 void my_square(int x)           // 定义一个函数
2 {
3     cout << x*x << endl;       // 函数的具体内容
4 }
5
6 auto pfunc = &my_square;        // 只能用指针去操作函数，指针不是函数
7 (*pfunc)(3);                   // 可以用*访问函数
8 pfunc(3);                      // 也可以直接调用函数指针
9
```

所以，在面向过程编程范式里，函数和变量虽然是程序里最关键的两个组成部分，但却因为没有值、没有作用域而不能一致地处理。函数只能是函数，变量只能是变量，彼此之间虽不能说是“势同水火”，但至少是“泾渭分明”。

认识 lambda

好了，搞清楚了函数，现在再来看看 C++11 引入的 lambda 表达式，下面是一个简单的例子：

 复制代码

```
1 auto func = [](int x)           // 定义一个lambda表达式
2 {
3     cout << x*x << endl;       // lambda表达式的具体内容
4 };
5
6 func(3);                       // 调用lambda表达式
```

暂时不考虑代码里面的语法细节，单从第一印象上，我们可以看到有一个函数，但更重要的，是这个函数采用了赋值的方式，存入了一个变量。

这就是 lambda 表达式与普通函数最大、也是最根本的区别。

因为 lambda 表达式是一个变量，所以，我们就可以“按需分配”，随时随地在调用点“**就地**”定义函数，限制它的作用域和生命周期，实现函数的局部化。

而且，因为 lambda 表达式和变量一样是“一等公民”，用起来也就更灵活自由，能对它做各种运算，生成新的函数。这就像是数学里的复合函数那样，把多个简单功能的小 lambda 表达式组合，变成一个复杂的大 lambda 表达式。

如果你比较熟悉 C++98，或者看过一些相关的资料，可能会觉得 lambda 表达式只不过是函数对象（function object）的一种简化形式，只是一个好用的“语法糖”（syntactic sugar）。


大道理上是没错的，但如果把它简单地等同于函数对象，认为它只是免去了手写函数对象的麻烦，那就实在是有点太“肤浅”了。

lambda 表达式为 C++ 带来的变化可以说是革命性的。虽然它表面上只是一个很小的改进，简化了函数的声明 / 定义，但深层次带来的编程理念的变化，却是非常巨大的。

这和 C++ 当初引入 bool、class、template 这些特性时有点类似，乍看上去好像只是一点点的语法改变，但后果却如同雪崩，促使人们更多地去思考、探索新的编程方向，而

lambda 引出的全新思维方式就是“函数式编程”——把写计算机程序看作是数学意义上的求解函数。

C++ 里的 lambda 表达式除了可以像普通函数那样被调用，还有一个普通函数所不具备的特殊本领，就是可以**“捕获”外部变量**，在内部的代码里直接操作。


 复制代码

```
1  int n = 10;                // 一个外部变量
2
3  auto func = [=](int x)      // lambda表达式，用“=”值捕获
4  {
5      cout << x*n << endl;    // 直接操作外部变量
6  };
7
8  func(3);                   // 调用lambda表达式
```

看到这里，如果你用过 JavaScript，那么一定会有种眼熟的感觉。没错，lambda 表达式就是在其他语言中大名鼎鼎的**“闭包”**（closure），这让它真正超越了函数和函数对象。

“闭包”是什么，很难一下子说清楚，我就不详细解释了。说得形象一点，你可以把闭包理解为一个“活的代码块”“活的函数”。它虽然在出现时被定义，但因为保存了定义时捕获的外部变量，就可以跳离定义点，把这段代码“打包”传递到其他地方去执行，而仅凭函数的入口参数是无法做到这一点的。

这就导致函数式编程与命令式编程（即面向过程）在结构上有很大不同，程序流程不再是按步骤执行的“死程序”，而是一个个的“活函数”，像做数学题那样逐步计算、推导出结果，有点像下面的这样：

 复制代码

```
1  auto a = [](int x)          // a函数执行一个功能
2      {...}
3  auto b = [](double x)       // b函数执行一个功能
4      {...}
5  auto c = [](string str)     // c函数执行一个功能
6      {...}
7
8  auto f = [...]              // f函数执行一个功能
9      {...}
10
11 return f(a, b, c)           // f调用a/b/c运算得到结果
```


你也可以再对比面向对象来理解。在面向对象编程里，程序是由一个个实体对象组成的，对象通信完成任务。而在函数式编程里，程序是由一个个函数组成的，函数互相嵌套、组合、调用完成任务。

不过，毕竟函数式编程在 C++ 里是一种较新的编程范式，而且面向过程里的函数概念“根深蒂固”，我说了这么多，你可能还是不太能领会它的奥妙，这也很正常。

下面我就来讲讲 lambda 表达式的使用细节，掌握了以后多用，就能够更好地理解了。

使用 lambda 的注意事项

要学好用好 lambda，我觉得就是三个重点：语法形式，变量捕获规则，还有泛型的用法。


1. lambda 的形式

首先你要知道，C++ 没有为 lambda 表达式引入新的关键字，并没有“lambda”这样的词汇，而是用了一个特殊的形式“[]”，术语叫“**lambda 引出符**”（lambda introducer）。

在 lambda 引出符后面，就可以像普通函数那样，用圆括号声明入口参数，用花括号定义函数体。

下面的代码展示了我最喜欢的一个 lambda 表达式（也是最简单的）：

```
1 auto f1 = [](){}; // 相当于空函数，什么也不做
```

 复制代码

这行语句定义了一个相当于空函数的 lambda 表达式，三个括号“排排坐”，看起来有种奇特的美感，让人不由得想起那句经典台词：“一家人最要紧的就是整整齐齐。”（不过还是差了个尖括号 <>）。

当然了，实际开发中不会有这么简单的 lambda 表达式，它的函数体里可能会有很多语句，所以**一定要有良好的缩进格式**——特别是有嵌套定义的时候，尽量让人能够一眼就看出 lambda 表达式的开始和结束，必要的时候可以用注释来强调。


 复制代码

```
1  auto f2 = []()                // 定义一个lambda表达式
2  {
3      cout << "lambda f2" << endl;
4
5      auto f3 = [](int x)        // 嵌套定义lambda表达式
6      {
7          return x*x;
8      }; // lambda f3            // 使用注释显式说明表达式结束
9
10     cout << f3(10) << endl;
11 }; // lambda f2                // 使用注释显式说明表达式结束
```

你可能注意到了，在 lambda 表达式赋值的时候，我总是使用 auto 来推导类型。这是因为，在 C++ 里，每个 lambda 表达式都会有一个独特的类型，而这个类型只有编译器才知道，我们是无法直接写出来的，所以必须用 auto。

不过，因为 lambda 表达式毕竟不是普通的变量，所以 C++ 也鼓励程序员**尽量“匿名”使用 lambda 表达式**。也就是说，它不必显式赋值给一个有名字的变量，直接声明就能用，免去你费力起名的烦恼。

这样不仅可以让代码更简洁，而且因为“匿名”，lambda 表达式调用完后也就不存在了（也有被拷贝保存的可能），这就最小化了它的影响范围，让代码更加安全。

 复制代码

```
1  vector<int> v = {3, 1, 8, 5, 0}; // 标准容器
2
3  cout << *find_if(begin(v), end(v), // 标准库里的查找算法
4              [](int x)              // 匿名lambda表达式，不需要auto赋值
5              {
6                  return x >= 5;      // 用做算法的谓词判断条件
7              }                      // lambda表达式结束
8              )
9      << endl;                      // 语句执行完，lambda表达式就不存在了
```

2.lambda 的变量捕获

lambda 的“捕获”功能需要在“[]”里做文章，由于实际的规则太多太细，记忆、理解的成本高，所以我只说几个要点，帮你快速掌握它们：

“[=]”表示按值捕获所有外部变量，表达式内部是值的拷贝，并且不能修改；

“[&]”是按引用捕获所有外部变量，内部以引用的方式使用，可以修改；

你也可以在“[]”里明确写出外部变量名，指定按值或者按引用捕获，C++ 在这里给予了非常大的灵活性。

 复制代码

```
1  int x = 33;           // 一个外部变量
2
3  auto f1 = [=]()       // lambda表达式，用“=”按值捕获
4  {
5      //x += 10;        // x只读，不允许修改
6  };
7
8  auto f2 = [&]()       // lambda表达式，用“&”按引用捕获
9  {
10     x += 10;          // x是引用，可以修改
11 };
12
13 auto f3 = [=, &x]()   // lambda表达式，用“&”按引用捕获x，其他的按值捕获
14 {
15     x += 20;          // x是引用，可以修改
16 };
```

“捕获”也是使用 lambda 表达式的一个难点，关键是要理解 “外部变量” 的含义。

我建议，你可以简单地按照其他语言的习惯，称之为 **“upvalue”**，也就是在 lambda 表达式定义之前所有出现的变量，不管它是局部的还是全局的。

这就有一个变量生命周期的问题。

使用 “[=]” 按值捕获的时候，lambda 表达式使用的是变量的独立副本，非常安全。而使用 “[&]” 的方式捕获引用就存在风险，当 lambda 表达式在离定义点“很远的地方”被调用的时候，引用的变量可能发生了变化，甚至可能会失效，导致难以预料的后果。

所以，我建议你在使用捕获功能的时候要小心，对于“就地”使用的小 lambda 表达式，可以用 “[&]” 来减少代码量，保持整洁；而对于非本地调用、生命周期较长的 lambda 表达式应慎用 “[&]” 捕获引用，而且，最好是在 “[]” 里显式写出变量列表，避免捕获不必要的变量。

复制代码

```
1 class DemoLambda final
2 {
3 private:
4     int x = 0;
5 public:
6     auto print()           // 返回一个lambda表达式供外部使用
7     {
8         return [this]()    // 显式捕获this指针
9         {
10             cout << "member = " << x << endl;
11         };
12     }
13 };
```

3. 泛型的 lambda

在 C++14 里，lambda 表达式又多了一项新本领，可以实现“泛型化”，相当于简化了的模板函数，具体语法还是利用了“多才多艺”的 auto：

复制代码

```
1 auto f = [](const auto& x)           // 参数使用auto声明，泛型化
2 {
3     return x + x;
4 };
5
6 cout << f(3) << endl;                // 参数类型是int
7 cout << f(0.618) << endl;            // 参数类型是double
8
9 string str = "matrix";
10 cout << f(str) << endl;             // 参数类型是string
```

这个新特性在写泛型函数的时候非常方便，摆脱了冗长的模板参数和函数参数列表。如果你愿意的话，可以尝试在今后的代码里都使用 lambda 来代替普通函数，能够少写很多代码。

小结

今天我讲了 lambda 表达式。它不仅是对旧有函数对象的简单升级，而是更高级的“闭包”，给 C++ 带来了新的编程理念：函数式编程范式。

在 C 语言里，函数是一个“静止”的代码块，只能被动地接受输入然后输出。而 lambda 的出现则让函数“活”了起来，极大地提升了函数的地位和灵活性。

比照“智能指针”的说法，lambda 完全可以称为是“智能函数”，价值体现在就地定义、变量捕获等能力上，它也给 C++ 的算法、并发（线程、协程）等后续发展方向铺平了道路，在后面讲标准库的时候，我们还会多次遇到它。

虽然目前在 C++ 里，纯函数式编程还比较少见，但“轻度”使用 lambda 表达式也能够改善代码，比如用“map+lambda”的方式来替换难以维护的 if/else/switch，可读性要比大量的分支语句好得多。

小结一下今天的要点内容：

1. lambda 表达式是一个闭包，能够像函数一样被调用，像变量一样被传递；
2. 可以使用 auto 自动推导类型存储 lambda 表达式，但 C++ 鼓励尽量就地匿名使用，缩小作用域；
3. lambda 表达式使用 “[=]” 的方式按值捕获，使用 “[&]” 的方式按引用捕获，空的 “[]” 则是无捕获（也就相当于普通函数）；
4. 捕获引用时必须要注意外部变量的生命周期，防止变量失效；
5. C++14 里可以使用泛型的 lambda 表达式，相当于简化的模板函数。

末了我再说一句，和 C++ 里的大多数新特性一样，滥用 lambda 表达式的话，就会产生一些难以阅读的代码，比如多个函数的嵌套和串联、调用层次过深。这也需要你在实践中慢慢积累经验，找到最适合你自己的使用方式。

课下作业

最后是课下作业时间，给你留两个思考题吧：

1. 你对函数式编程有什么样的理解和认识呢？
2. lambda 表达式的形式非常简洁，可以在很多地方代替普通函数，那它能不能代替类的成员函数呢？为什么？

欢迎你在留言区写下你的思考和答案，如果觉得今天的内容对你有所帮助，也欢迎分享给你的朋友。我们下节课见。

课外小贴士

1. 目前，lambda表达式还不支持function-try，只能在函数体内部用try-catch。
2. lambda表达式的返回值类型可以自动推导（相当于用了auto），但有的时候必须明确指定返回值类型，这个时候就得用比较“怪异”的返回值后置语法，在入口参数的圆括号后用“-> type”的形式。
3. 在按值捕获外部变量的时候，可以给lambda表达式加上mutable修饰，允许修改变量。注意，这与按引用捕获不同，修改的只是变量的拷贝，不影响外部变量的原值。
4. 如果确实需要长期持有外部变量，为了避免变量失效，可以考虑使用shared_ptr。

5. 因为每个lambda表达式的类型都是唯一的，所以即使函数签名相同，lambda变量也不能互相赋值。解决办法是使用标准库里的std::function类，它是“函数的容器”“智能函数指针”，可以存储任意符合签名的“可调用物”（callable object），搭配使用能够让lambda表达式用起来更灵活。



课程预告

6月-7月课表抢先看

充 ¥500 得 ¥580

赠「¥ 118 月球主题 AR 笔记本」



【点击】图片，立即查看 >>>

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 09 | exception：怎样才能用好异常？

下一篇 11 | 一枝独秀的字符串：C++也能处理文本？

精选留言 (16)

写留言



罗剑锋 置顶

2020-05-29

我在GitHub的lambd.cpp里写了一小段代码，示范了function + lambda实现成员函数的方法，算是对课下作业2的一个参考，同学们可以看看。

展开 ▾



2



冻冻

2020-05-28

老师，用“map+lambda”的方式来替换难以维护的 if/else/switch，能举个例子吗？

作者回复: 这个需要用到std::function，存储lambda表达式，比如

~~~

```
map<int, function<void()>> funcs;
```

```
funcs[1] = [](){...};
```

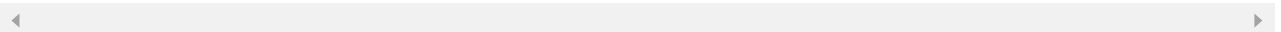
```
funcs[7] = [](){...};
```

```
funcs[42] = [](){...};
```

```
return funcs[x]();
```

~~~

这样，就把switch/case语句转换成了function+lambda，让map替你自动switch。



1

3



被讨厌的勇气

2020-05-28

采用lambda表达式替换类的成员函数，成员变量通过 '[this]'可以捕获（相当于成员函数中的this参数），参数、返回值、函数体，lambda表达式都可以实现，所以理论上，是可以替换的。

试了一下，报错：在类内部无法定义auto。之前老师提到过的。

展开 ▾

作者回复: 对的，就是这个原因。





2



this_is_for_u

2020-05-28

个人认为lambda表达式还有个重要的用途是它可以自定义stl函数谓词规则(pred)，例如自定义排序规则，而无需使用传统的仿函数那种麻烦的方法。

展开

作者回复: 对，lambda大大方便了算法还有并发，改变了C++的编程方式。



4

1



完全不会C++

2020-05-28

老师早啊!

展开

作者回复: morning



1



Eason Tai

2020-05-30

我理解lambda可以炫技就很帅，减少冗余。不过，一定要适度。

lambda 表达式的形式非常简洁，可以在很多地方代替普通函数，那它能不能代替类的成员函数呢？为什么？

...

展开

作者回复: 回答的不太正确，原因是auto在类里的限制，可以参考其他同学的回答。

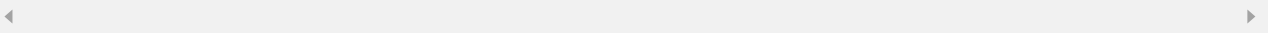


Eason Tai

2020-05-30

创建lambda函数等于创建了新的类型，对操作符 () 进行了重载。所以直接 `auto f = [=] ()` 相当于对一个未初始化成员函数进行auto类型定义，所以报错。这样理解对吗？

作者回复: 差不多, 可以再看第6讲, lambda表达式赋值必须用auto, 但auto不能用在类成员初始化。



1



Tedeer

2020-05-29

我看到老师文章中说到每个lambda表达式都有个全局唯一类型, 只有编译器知道; lambda表达式只能通过auto声明, 且auto变量必须在定义时初始化, 而在类声明时, 成员并未被赋值, 就不知道lambda表达式类型, 无法推导出具体类型, 编译器会报错, 所以无法使用lambda表达式作成员函数, 请问老师我这样理解对吗?

展开 ▾

作者回复: 理解的差不多, 可以参考一下auto那讲, 因为目前C++不允许在声明成员变量时用auto推导类型。

也可以再试着写一下代码来验证。



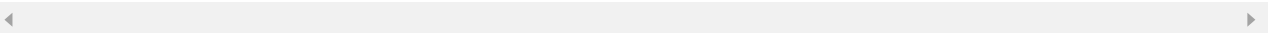
tt

2020-05-29

对lambda的意义, 老师这一讲真是拨云见雾啊。相对于functor, 它给了我们一个理解世界、表达世界的新角度

展开 ▾

作者回复: lambda超越了早期的函数对象, 因为它是“闭包”, 所以有着与函数、函数对象完全不同的用法, 可以说是一种“高维生物”。



黄骏

2020-05-29

之前觉得用lambda可能是炫技, 现在看来它也有特定的合适的应用场景, 可能接触函数式编程太久, 没转过弯来

展开 ▾

作者回复: 任何C++特性都能被用来炫技, 只是lambda实在是太炫了, 刚学会难免忍不住炫一下, 完全可以理解, 这也是人之常情。

用多了才会真正领会它的精神, 做到大象无形。



LDxy

2020-05-28

auto pfunc = &my_square; // 只能用指针去操作函数, 指针不是函数
(*pfunc)(3); // 调用需要用*, 才能访问函数

如果写成pfunc(3)来调用函数对不对呢?

展开 v

作者回复: 也是可以, 我写的时候有点思维混乱了, 感谢指正。

你也可以实际写代码试试, 让编译器告诉你答案。



EncodedStar

2020-05-28

如果把lambda 表达式看做变量, 比如文章中的: " auto f3 = [](int x) {return x*x; }", 我感觉是能做成员函数的, 相当于成员变量一样使用, 不知道这样理解有没有错

作者回复: 试着写一下代码, 看看会是什么样。



张JL

2020-05-28

我常用lambda替换函数中的小段重复代码。

相同代码重复写几遍感觉很蠢, 拿出来做成函数又没有必要, 因为没有其他的调用需求, 这时候用lambda就简洁多了

作者回复: 对, 用lambda来封装小段代码, 这就有点函数式编程的意思了。



无为而立

2020-05-28

在python中使用过lambda map filter，感觉lambda可以做一些简单运算，特别是纯数学的计算。不宜太多使用。不然阅读起来费劲儿

展开 ∨

作者回复: python 的lambda 我感觉比较笨拙，而C++的lambda 轻巧强大，应该多用，后面可以看到更多的示范用法，可以参考一下。



java2c++

2020-05-28

问题1个人见解：我是Java技术栈，最近在看老师的c++课程，lambda现在Java也有了，在Java中可以简化匿名内部类的调用写法，可以用于策略设计模式

展开 ∨

作者回复: 多种语言互相借鉴，可以让我们写出更好的代码。



java2c++

2020-05-28

lambda不能代替成员函数的原因我猜测是作用域的问题，lambda只能在函数内调用，总不能把所有的逻辑都写到main函数里吧

展开 ∨

作者回复: 不是这个原因，可以自己写代码试试看，实践一下。

