C ++ Lambda表达式详解

原创 奥修的灵魂 2020-12-19 14:08:21 2430 收藏 25 版权
分类专栏： C++学习

C ++ Lambda表达式详解

1.Lambda表达式概述

Lambda表达式是现代C++在C ++ 11和更高版本中的一个新的语法糖 ，在C++11、C++14、C++17和C++20中Lambda表达的内容还在不断更新。 lambda表达式（也称为lambda函数）是在调用或作为函数参数传递的位置处定义匿名函数对象的便捷方法。通常，lambda用于封装传递给算法或异步方法的几行代码 。本文主要介绍Lambda的工作原理以及使用方法。

2.Lambda表达式定义

2.1 Lambda表达式示例

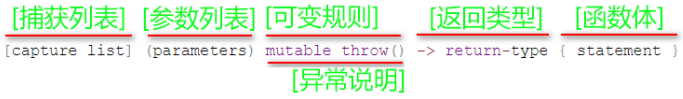
Lambda有很多叫法，有Lambda表达式、Lambda函数、匿名函数，本文中为了方便表述统一用Lambda表达式进行叙述。 ISO C ++标准官网展示了一个简单的lambda 表示式实例:

```
#include <algorithm>
#include <cmath>

void abssort(float* x, unsigned n) {
    std::sort(x, x + n,
        // Lambda expression begins
        [](float a, float b) {
            return (std::abs(a) < std::abs(b));
        } // end of lambda expression
    );
}
```

在上面的实例中std::sort函数第三个参数应该是传递一个排序规则的函数，但是这个实例中直接将排序函数的实现写在应该传递函数的位置，省去了定义排序函数的过程，对于这种不需要复用，且短小的函数，直接传递函数体可以增加代码的可读性。

2.2 Lambda表达式语法定义



1. 捕获列表。在C ++规范中也称为*Lambda*导入器， 捕获列表总是出现在*Lambda*函数的开始处。实际上，*[]*是*Lambda*引出符。编译器根据该引出符判断接下来的代码是否是*Lambda*函数，捕获列表能够捕捉上下文中的变量以供*Lambda*函数使用。
2. 参数列表。与普通函数的参数列表一致。如果不需要参数传递，则可以连同括号“()”一起省略。
3. 可变规格*。mutable修饰符， 默认情况下Lambda函数总是一个const函数，mutable可以取消其常量性。在使用该修饰符时，参数列表不可省略（即使参数为空）。*
4. 异常说明。用于Lamdba表达式内部函数抛出异常。
5. 返回类型。 追踪返回类型形式声明函数的返回类型同符号”->”一起省略。此外，在返回类型明确的情况

型进行推导。

6. lambda函数体。内容与普通函数一样，不过除了可以使用参数之外，还可以使用所有捕获的变量。

2.3 Lambda表达式参数详解

2.3.1 Lambda捕获列表

Lambda表达式与普通函数最大的区别是，除了可以使用参数以外，Lambda函数还可以通过捕获列表访问一些上下文中的数据。具体地，捕捉列表描述了上下文中哪些数据可以被Lambda使用，以及使用方式（以值传递的方式或引用传递的方式）。语法上，在“[]”包括起来的是捕获列表，捕获列表由多个捕获项组成，并以逗号分隔。捕获列表有以下几种形式：

- []表示不捕获任何变量

```
auto function = ([]{
        std::cout << "Hello World!" << std::endl;
    }
);

function();
```

- [var]表示值传递方式捕获变量var

```
int num = 100;
auto function = ([num]{
        std::cout << num << std::endl;
    }
);

function();
```

- [=]表示值传递方式捕获所有父作用域的变量（包括this）

```
int index = 1;
int num = 100;
auto function = ([=]{
            std::cout << "index: "<< index << ", "
                << "num: "<< num << std::endl;
    }
);

function();
```

- [&var]表示引用传递捕捉变量var

```
int num = 100;
auto function = ([&num]{
        num = 1000;
        std::cout << "num: " << num << std::endl;
    }
);

function();
```

- [&]表示引用传递方式捕捉所有父作用域的变量（包括this）

```
int index = 1;
int num = 100;
auto function = ([&]{
        num = 1000;
        index = 2;
        std::cout << "index: "<< index << ", "
            << "num: "<< num << std::
    }
```

```
);

function();
```

- [this]表示值传递方式捕捉当前的this指针

```
#include <iostream>
using namespace std;

class Lambda
{
public:
    void sayHello() {
        std::cout << "Hello" << std::endl;
    };

    void lambda() {
        auto function = [this]{
            this->sayHello();
        };

        function();
    }
};

int main()
{
    Lambda demo;
    demo.lambda();
}
```

- [=, &] 拷贝与引用混合

  - [=,&a,&b]表示以引用传递的方式捕捉变量a和b，以值传递方式捕捉其它所有变量。

```
int index = 1;
int num = 100;
auto function = ([=, &index, &num]{
        num = 1000;
        index = 2;
        std::cout << "index: "<< index << ", "
            << "num: "<< num << std::endl;
    }
);

function();
```

```

```

- [&,a,this]表示以值传递的方式捕捉变量a和this，引用传递方式捕捉其它所有变量。

不过值得注意的是，捕捉列表不允许变量重复传递。下面一些例子就是典型的重复，会导致编译时期的错误。例如：

- [=,a]这里已经以值传递方式捕捉了所有变量，但是重复捕捉a了，会报错的;
- [&,&this]这里&已经以引用传递方式捕捉了所有变量，再捕捉this也是一种重复。

如果lambda主体 `total` 通过引用访问外部变量，并 `factor` 通过值访问外部变量，则以下捕获子句是等效的：

```
[&total, factor]
[factor, &total]
[&, factor]
[factor, &]
```

```
[=, &total]
[&total, =]
```

2.3.2 Lambda参数列表

除了捕获列表之外，lambda还可以接受输入参数。参数列表是可选的，并且在大多数方面类似于函数的参数列表。

```
auto function = [] (int first, int second){
    return first + second;
};

function(100, 200);
```

2.3.3 可变规格mutable

mutable修饰符， 默认情况下Lambda函数总是一个const函数，mutable可以取消其常量性。在使用该修饰符时，参数列表不可省略（即使参数为空）。

```
#include <iostream>
using namespace std;

int main()
{
   int m = 0;
   int n = 0;
   [&, n] (int a) mutable { m = ++n + a; }(4);
   cout << m << endl << n << endl;
}
```

2.3.4 异常说明

你可以使用 throw() 异常规范来指示 lambda 表达式不会引发任何异常。与普通函数一样，如果 lambda 表达式声明 C4297 异常规范且 lambda 体引发异常，Visual C++ 编译器将生成警告 throw()。

```
int main() // C4297 expected
{
    []() throw() { throw 5; }();
}
```

在MSDN的异常规范 [5] 中，明确指出异常规范是在 C++11 中弃用的 C++ 语言功能。因此这里不建议不建议大家使用。

2.3.5 返回类型

Lambda表达式的返回类型会自动推导。除非你指定了返回类型，否则不必使用关键字。返回型类似于通常的方法或函数的返回型部分。但是，返回类型必须在参数列表之后，并且必须在返回类型->之前包含类型关键字。如果lambda主体仅包含一个return语句或该表达式未返回值，则可以省略Lambda表达式的return-type部分。如果lambda主体包含一个return语句，则编译器将从return表达式的类型中推断出return类型。否则，编译器将返回类型推导为void。

```
auto x1 = [](int i){ return i; };
```

2.3.6 Lambda函数体

Lambda表达式的lambda主体（标准语法中的复合语句）可以包含普通方法或函数的主体可以包含的任何内容。普通函数和lambda表达式的主体都可以访问以下类型的变量：

```
- 捕获变量
- 形参变量
- 局部声明的变量
- 类数据成员，当在类内声明**`this`**并被捕获时
- 具有静态存储持续时间的任何变量，例如全局变量
```

```cpp
#include <iostream>
using namespace std;

int main()
{
   int m = 0;
   int n = 0;
   [&, n] (int a) mutable { m = ++n + a; }(4);
   cout << m << endl << n << endl;
}
```

3.Lambda表达式的优缺点

3.1 Lambda表达式的优点

- 可以直接在需要调用函数的位置定义短小精悍的函数，而不需要预先定义好函数

```cpp
std::find_if(v.begin(), v.end(), [](int& item){return item > 2});
```

- 使用Lamdba表达式变得更加紧凑，结构层次更加明显、代码可读性更好

3.2 Lambda表达式的缺点

- Lamdba表达式语法比较灵活，增加了阅读代码的难度
- 对于函数复用无能为力

4.Lambda表达式工作原理

4.1 Lambda表达式工作原理

编译器会把一个lambda表达式生成一个匿名类的匿名对象，并在类中重载函数调用运算符,实现了一个operator()方法。

```cpp
auto print = [](){cout << "Hello World!" << endl; };
```

编译器会把上面这一句翻译为下面的代码:

```cpp
class print_class
{
public:
    void operator()(void) const
    {
        cout << "Hello World! " << endl;
    }
};
//用构造的类创建对象，print此时就是一个函数对象
auto print = print_class();
```

4.2 C++仿函数

仿函数（functor）又称为函数对象（function object）是一个能行使函数功能的类。仿函数的语法几乎和我们普通的函数调用一样，不过作为仿函数的类，都必须重载operator()运算符，仿函数与Lamdba表达式的作用是一致的。举个例子:

```cpp
#include <iostream>
#include <string>
using namespace std;

class Functor
{
public:
    void operator() (const string& str) const
    {
        cout << str << endl;
```

```
    }
};

int main()
{
    Functor myFunctor;
    myFunctor("Hello world!");
    return 0;
}
```

5.Lamdba表达式适用场景

5.1 Lamdba表达式应用于STL算法库

```
// for_each应用实例
int a[4] = {11, 2, 33, 4};
sort(a, a+4, [=](int x, int y) -> bool { return x%10 < y%10; } );
for_each(a, a+4, [=](int x) { cout << x << " ";} );
```

```
// find_if应用实例
int x = 5;
int y = 10;
deque<int> coll = { 1, 3, 19, 5, 13, 7, 11, 2, 17 };
auto pos = find_if(coll.cbegin(), coll.cend(), [=](int i) {
    return i > x && i < y;
});
```

```
// remove_if应用实例
std::vector<int> vec_data = {1, 2, 3, 4, 5, 6, 7, 8, 9};
int x = 5;
vec_data.erase(std::remove_if(vec.date.begin(), vec_data.end(), [](int i) {
    return n < x;}), vec_data.end());

std::for_each(vec.date.begin(), vec_data.end(), [](int i) {
    std::cout << i << std::endl;});
```

5.2 短小不需要复用函数场景

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main(void)
{
    int data[6] = { 3, 4, 12, 2, 1, 6 };
    vector<int> testdata;
    testdata.insert(testdata.begin(), data, data + 6);

    // 对于比较大小的逻辑，使用lamdba不需要在重新定义一个函数
    sort(testdata.begin(), testdata.end(), [](int a, int b){
        return a > b; });

    return 0;
}
```

5.3 Lamdba表达式应用于多线程场景

```
#include <iostream>
#include <thread>
#include <vector>
#include <algorithm>

int main()
{
    // vector 容器存储线程
```

```
    std::vector<std::thread> workers;
    for (int i = 0; i < 5; i++)
    {
        workers.push_back(std::thread([]()
        {
            std::cout << "thread function\n";
        }));
    }
    std::cout << "main thread\n";

    // 通过 for_each 循环每一个线程
    // 第三个参数赋值一个task任务
    // 符号'[]'会告诉编译器我们正在用一个匿名函数
    // lambda函数将它的参数作为线程的引用t
    // 然后一个一个的join
    std::for_each(workers.begin(), workers.end(), [](std::thread &t;)
    {
        t.join();
    });

    return 0;
}
```

```
std::mutex mutex;
std::condition_variable condition;
std::queue<std::string> queue_data;

std::thread threadBody([&]{
    std::unique_lock<std::mutex> lock_log(mutex);
    condition.wait(lock_log, [&]{
        return !queue_data.front();
    });
    std::cout << "queue data: " << queue_data.front();
    lock_log.unlock();
});

queue_data.push("this is my data");
condition.notity_one();

if(threadBody.joinable())
{
    threadBody.join();
}
```

5.4 Lamdba表达式应用于函数指针与function

```
#include <iostream>
#include <functional>
using namespace std;

int main(void)
{
    int x = 8, y = 9;
    auto add = [](int a, int b) { return a + b; };
    std::function<int(int, int)> Add = [=](int a, int b) { return a + b; };

    cout << "add: " << add(x, y) << endl;
    cout << "Add: " << Add(x, y) << endl;

    return 0;
}
```

5.5 Lamdba表达式作为函数的入参

```
using FuncCallback = std::function<void(void)>;

void DataCallback(FuncCallback callback)
{
```

```
    std::cout << "Start FuncCallback!" << std::endl;
    callback();
    std::cout << "End FuncCallback!" << std::endl;
}

auto callback_handler = [&](){
    std::cout << "This is callback_handler";
};

DataCallback(callback_handler);
```

5.6 Lamdba表达式在QT中的应用

```
QTimer *timer=new QTimer;
timer->start(1000);
QObject::connect(timer,&QTimer::timeout,[&](){
        qDebug() << "Lambda表达式";
});
```

```
int a = 10;
QString str1 = "汉字博大精深";
connect(pBtn4, &QPushButton::clicked, [=](bool checked){
    qDebug() << a <<str1;
    qDebug() << checked;
    qDebug() << "Hua Windows Lambda Button";
});
```

6.总结

对于Lambda这种新东西，有的人用的非常爽，而有的人看着都不爽。仁者见仁，智者见智。不管怎么样，作为程序员的你，都要会的。这篇文章就是用来弥补自己对C++ Lambda表达式的认知不足的总结，在很多开源代码中Lamdba表达式使用的非常多，如果不深入学习Lamdba相关知识，在未来学习开源代码就会有很多障碍。


```
timer->start(1000);
QObject::connect(timer,&QTimer::timeout,[&](){
        qDebug() << "Lambda表达式";
});
```

```
int a = 10;
QString str1 = "汉字博大精深";
connect(pBtn4, &QPushButton::clicked, [=](bool checked){
    qDebug() << a <<str1;
    qDebug() << checked;
    qDebug() << "Hua Windows Lambda Button";
});
```

6.总结

对于Lambda这种新东西，有的人用的非常爽，而有的人看着都不爽。仁者见仁，智者见智。不管怎么样，作为程序员的你，都要会的。这篇文章就是用来弥补自己对C++ Lambda表达式的认知不足的总结，在很多开源代码中Lamdba表达式使用的非常多，如果不深入学习Lamdba相关知识，在未来学习开源代码就会有很多障碍。

和苏Hibna： 针不戳 4 月前 回复

qq_44183732： 真不错 6 月前 回复

C++Lambda表达式_alex1997222的博客 10-17
Lambda表达式可以使用其可见范围内的外部变量,但必须明确声明(明确声明哪些外部变量可以被该Lambda表达式…

C++ lambda表达式 (一)_heshaai6843的博客 9-22
C++ lambda表达式 (一) 为什么要lambda函数 匿名函数是许多编程语言都支持的概念,有函数体,没有函数名。195…

Android 中Lambda表达式的使用实例详解 08-30
主要介绍了 Android 中Lambda表达式的使用实例详解的相关资料,需要的朋友可以参考下

C++ 中的 Lambda 表达式 10-29
C++11中才支持lambda表达式，使用lambda，可以直接代替回调函数。不需要再去写一个函数，直接在调用的地…

c++中lambda表达式的用法_小白进阶的博客_c++ lamda用法 10-23
c++中lambda表达式的用法 1.基础用法 c++11提供了对匿名函数的支持,称为Lambda函数(也叫Lambda表达式)。L…

串口文件传输系统源代码 01-13
一款串口文件传输系统的源代码。

C++ Lambda表达式详解 请叫我皮皮虾的博客 1万+
转自：https://blog.csdn.net/u010984552/article/details/53634513 一段简单的Code 我也不是文艺的人，对于Lam…

【C++11新特性】 Lambda表达式 上善若水，人淡如菊 3895
C++11的一大亮点就是引入了Lambda表达式。利用Lambda表达式，可以方便的定义和创建匿名函数。对于C++…

C++中的lambda表达式，这样学习就对了！ ISmileLi的博客 5748
C++中的lambda表达式零、小序一、lambda表达式介绍1、lambda表达式概念2、lambda表达式的优势3、lambd…

C++中的Lambda表达式详解 热门推荐 NickWei的博客 3万+
一直都在提醒自己，我是搞C++的；但是当C++11出来这么长时间了，我却没有跟着队伍走，发现很对不起自己的…

详解C++的lambda表达式 码之有道 822
1.概述 C++ 11 中的 Lambda 表达式用于定义并创建匿名的函数对象，以简化编程工作。 Lambda 的语法形式如…

java8多核编程++ lambda表达式 04-28
《精通lambda表达式：Java多核编程》介绍Java SE 8中与lambda相关的特性是如何帮助Java迎接下一代并行硬…

C ++文件读写 08-08
C++文件读写的相关操作，可以直接使用,适合初学者

C# lambda表达式原理定义及实例详解 08-18
主要介绍了C# lambda表达式原理定义及实例详解,文中通过示例代码介绍的非常详细，对大家的学习或者工作具…

Mac OS Big Sur主题 08-11
在Windows上安装Mac OS Big Sur主题视频教程以及所需文件！ 其中包括：Mac OS Big Sur主题（黑色+白色+白…

Java8 Lambda表达式详解及实例 最新发布 09-01
主要介绍了Java8 Lambda表达式详解的相关资料,需要的朋友可以参考下

C++ Lambda 函数与表达式 weixin_41521681的博客 72
Lambda 函数与表达式 C++11 提供了对匿名函数的支持,称为 Lambda 函数(也叫 Lambda 表达式)。 Lambda 表达…

Lambda 函数与表达式 weixin_33701617的博客 68
2019独角兽企业重金招聘Python工程师标准>>> ...

什么是C ++ 11中的lambda表达式? asdfgh0077的博客 979
What is a lambda expression in C++11? 什么是C ++ 11中的lambda表达式? When would I use one? 我什么时候…

C++ lambda函数详解 飞行的荷兰猪 3785
c++中lambda函数是作为c++11新特新添加到c++中的，其主要是以匿名函数捕获scope内变量的方式构造闭包（cl…

奥修的灵魂
码龄11年 暂无认证

1	50万+	115万+	2959	
原创	周排名	总排名	访问	等级

21	2	6	4	25
积分	粉丝	获赞	评论	收藏

私信 关注

搜博主文章

热门文章

C ++ Lambda表达式详解 2412

分类专栏

C#编程
Ado.Net编程
WPF编程
C++学习 1篇
C语言学习
Linux随笔

最新评论

C ++ Lambda表达式详解
奥修的灵魂: 已修复

C ++ Lambda表达式详解
whiteyetihw: 2.2的图崩了

C ++ Lambda表达式详解
和苏Hibna: 针不戳

C ++ Lambda表达式详解
qq_44183732: 真不错

您愿意向朋友推荐“博客详情页”吗?

强烈不推荐 不推荐 一般般 推荐 强烈推荐

最新文章

2020年 1篇

目录