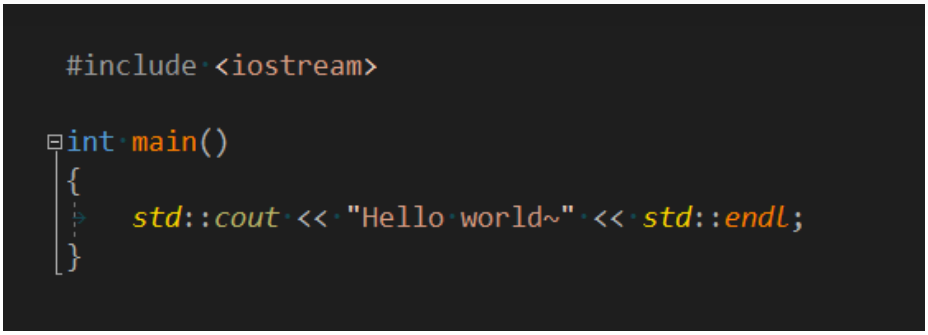

C++随笔：lambda 表达式

问尘
UE4学习中/图形学爱好者
+ 关注他

➤ lambda 表达式

是C++11引入的一个**语法糖**，可以快速的创建函数对象，既然是一个对象，那 lambda 本质上其实就**是一个类**，这个类**重载**了圆括号操作符 **operate()** 使其行为像是一个函数，并且将**捕获列表中的数据作为私有成员**，在构造函数的时候，**作为构造函数初始化列表**。

lambda 表达式结构（作为一个表达式，不可遗漏末尾大括号后的分号）：

```
[Capture List] (Params) mutable-Optional constexpr-Optional exception-C
```

lambda 表达式结构含义：

① **Capture List**：**捕获列表**，不可省略：

```
int a = 666, b = 888;

//  (1) 没有捕获变量
[]() {};
//  (2) 按值捕获变量a，复制了a的值，是默认捕获方式，按值捕获的变量不允许修改，除非添
[a]() {}; // 内部不能修改变量a
[a]() mutable {}; // 内部可以修改变量a，但不会影响到外部同名变量，本质是值复制
//  (3) 按引用捕获变量a，作为变量a的引用，不需要额外选项即可修改，修改会同步到外部同
[&a]() {};
//  (4) 不同变量按不同方式捕获
[a, &b]() {}; // 变量a按照值捕获，变量b按照引用捕获
//  (5) 隐式捕获，外部所有变量都按值或按引用捕获，不建议直接对所有变量使用隐式捕获
[=]() {}; // 外部变量都按值捕获
[&]() {}; // 外部变量都按引用捕获
[=, &]() {}; // 编译报错：隐式捕获不能同时使用
//  (6) 隐式捕获和显示捕获搭配使用
[=, &a]() {}; // 除变量a显示的按引用捕获，其余外部变量都隐式按值捕获
```



```
//  (8) 捕获表达式（也可以通过捕获表达式来捕获单独的类成员）：
```

```
auto func1 = [c = a, &d = b]() { // 相当于给变量a和b取别名，用于函数体作用域，
    cout << "c = a = " << c << "\td = b = " << d << endl;
};
auto func2 = [c = b - a]() {  // 执行表达式 b - a 并将结果存于变量 c，用于函
    cout << "c = b - a = " << c << endl;
};
func1();
func2();
```

② **Params**：**参数列表**，无参时可以省略，但**省略后**捕获列表和函数体之间**不能有可选选项存在**：

```
// 参数为空，且没有可选选项或返回值时，可省略
[] { };

// 编译报错：省略参数列表后，跟了可选选项（函数返回值也是可选选项）
[] mutable { };
[] -> int { return 233; };
```

③ **mutable**：**可选关键字**，使得函数体内可以修改按值捕获的变量：

```
int num = 666;
[num]() { num = 888; }; // 编译报错，不可修改按值捕获的变量
[num]() mutable { num = 888; }; // 修改num不会改变外部同名变量，按值捕获本质是
```

④ **constexpr**: **可选关键字**，**常量表达式**，从C++17开始，lambda 支持声明为常量表达式/常量函数；

⑤ **exception**：**可选**，声明 lambda 表达式可以**抛出异常**；

⑥ **attribute**：**可选**，声明时执行 lambda 表达式**特性（属性）**；

⑦ **->RetType**：**可选**，**函数返回类型**，一般不直接写出，由函数体内 return 表达式来**推导**返回类型；

⑧ **body**：函数/表达式主体，不可省略；

补充：lambda 因为重载了 operate()，可以在声明时直接调用，看起来没什么用，一般用于优化代码的书写，使代码不那么臃肿：

```
[] () {
    cout << "\n\nHello World~\n\n";
}(); // 声明后立即执行
```

捕获变量的作用域：如果按**引用捕获了局部变量**，那 lambda 表达式就只能在局部作用域执行了，若保存其指针在别处调用，因为离开了之前的局部作用域，引用的**局部变量最终会被析构**，当按引用捕获局部变量后，表达式的执行必须**与按引用捕获的变量在同一作用域**，或者**深拷贝一个对象**（这里需要**特别注意**的是按引用捕获的局部变量，在离开作用域后可能还可以访问到，是因为在离开变量所在作用域后到执行 lambda 表达式之时，该局部变量并未析构，**离开作用域的变量销毁释放不一定时立刻执行的**）：

```
class MyClass
{
public:
```

```
    int *num;
    MyClass(int _num) { num = new int(_num); }
    MyClass(const MyClass& copyOb) { num = new int(*(copyOb.num)); }
    ~MyClass() { delete num; }
};

auto GetFunc1()
{
    MyClass ob = MyClass(666);
    return [&ob]() { cout << *(ob.num) << endl; };
}

auto GetFunc2()
{
    MyClass ob = MyClass(888);
    // 调用自定义拷贝构造函数，深拷贝一个对象
    return [copyOb = MyClass(ob)]() { cout << *(copyOb .num) << endl; }
}

int main()
{
    auto func1 = GetFunc1();
    // 输出一个未定义的int类型值，按引用捕获了GetFunc函数中的局部对象ob，
    // 回到main函数时ob已被析构，num指针已被释放，这里只能按值捕获
    func1();

    // 深拷贝了局部对象，使得数据得以复制保留
    auto func2 = GetFunc2(); // 输出 888
    func2();
    return 0;
}
```

点击进入：B站个人主页

点击进入：知乎个人主页

点击进入：公众号话题

微信公众号搜索：问尘

发布于 2021-11-02 09:22

C++ C++ 编程 Lambda 表达式

文章被以下专栏收录

C++随笔
我爱C++

推荐阅读

基础篇：Lambda 表达式和函数对象

以下文章给出了很棒的总结，我如下的总结主要来源于Microsoft的C++文档和 欧长坤的书。lambda表达式与函数对象目录： Lambda表达式 - 捕获列表 - 值捕获 - 引用捕获 - this指针 - 泛型捕获 ...

Fei D... 发表于Moder...

现代 C++：Lambda 表达式

Lambda 表达式（Lambda Expression）是 C++11 引入的一个“语法糖”，可以方便快捷地创建一个“函数对象”。 从 C++11 开始，C++ 有三种方式可以创建/传递一个可以被调用的对象： 函数指针...

FOCUS 发表于C++

C++中的Lambda表达式探究

世间事无常 发表于C++

C++ Lambda表达式的完整介绍

c++在c++11标准中引入了lambda表达式，一般用于定义匿名函数，使得代码更加灵活简洁。lambda表达式与普通函数类似，也有参数列表、返回值类型和函数体，只是它的定义方式更简洁，并且可以在...

冲冲

还没有评论

写下你的评论...