C++11 匿名函数

虚幻四高级程序开发工程师班

01 概述

匿名，表示不没有明确的名称。对于操作动作来说，操作对象名称是我们找到对象的唯一依据，也是我们能够更好的和计算机沟通的凭证。在常规操作中，我们调用函数，操作变量都需要使用名称。这也是一个基本前提。

在新的C++11标准中，加入了匿名函数，也称作Lambda函数（Lambda表达式），有时也称作闭包（Closures）。他允许你将一段函数逻辑直接加入到程序段落，并且是以片段方式存在。匿名函数加入弥补了长久以来C++无法灵活编写函数的目的。这意味着你可以将一个内联写在代码函数里（通常被传递给另一个函数，与函数对象（Functor）和函数指针有相似的概念）。

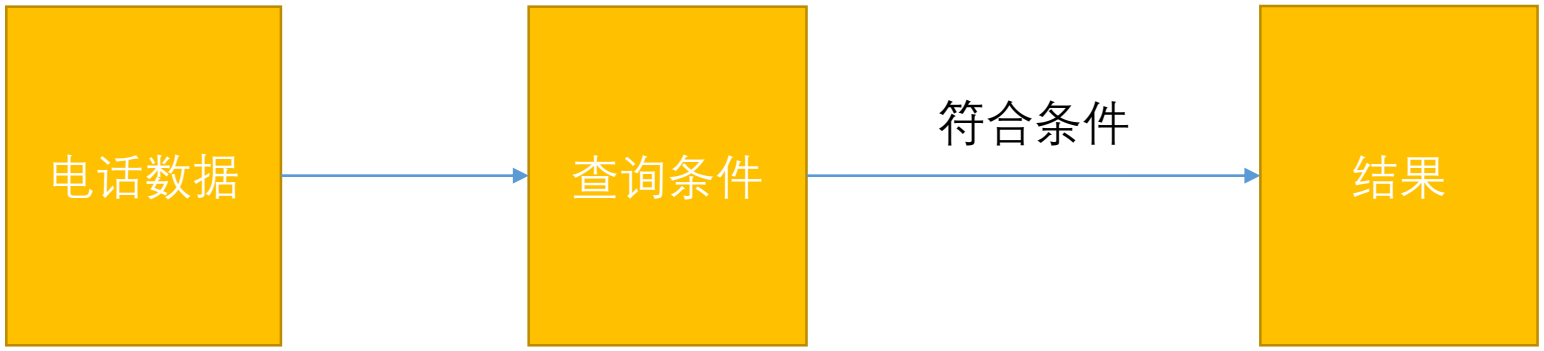
注意：匿名函数可以帮助我们在设计程序时解决一些特定问题，但也不是绝对必须需要使用的。所有的高阶语法均是为了快速生产而加入，高阶语法逻辑可以使用基本语法实现。

只有在新特性编译版本中使用。允许在编写函数时，不提供函数的名称，通过给定格式编写函数段，用以适配函数调用逻辑，或是参数操作。

匿名函数，可以可以降低函数编写设计时，一些函数只会被用到某一个功能点，无法具有宽泛的适用性，而设计的。

为什么会有匿名函数?

假设一个场景，我们存在一个地址簿。对外我们提供了一个搜寻方法，程序设计时，我们需要尽量多的考虑用户（使用者）对于搜寻的需求（**有些人希望按照姓名搜寻，有些人希望按照性别搜寻，有些人希望按照籍贯搜寻，有些人希望按照年龄搜寻……**）。我们发现，不同的用户在进行搜寻时，可能会有千奇百怪的搜索条件，但是对于我们来说，（系统设计者）只是希望知道当前数据条目是否复合查询条件，符合则返回，反之继续检索。那么我们在设计系统时，就无法面面俱到把所有查询条件均罗列出来，这就比较麻烦了。



为什么会有匿名函数?

我们是否可以把查询条件的构成，让查询人自己提供，这样可以大大提升查询的灵活性。因为查询者知道自己希望查询的逻辑是如何的，我们只是使用查询者提供的查询逻辑即可完成查询工作。

这样的查询方法，只会适用于某个场景，而不具备广泛的适用性，我们如果独立设计为函数，就有些得不偿失了。那么灵活的匿名函数可以帮助我们解决这样的问题了。

02 匿名函数

首先我们需要明白，匿名函数依旧是函数，他具备函数的所有特性。了解下匿名函数的表达式

[capture](parameters)->return-type{body;}

Capture：捕获外部变量访问方式说明符（可留白）

Parameters：参数表（如果没有则可以省略括号）

Return-type：返回类型（无返回类型则可以省略箭头和返回类型，或是通过隐式返回也可以省略箭头和返回类型）

Body：函数逻辑语句

03 基本操作

无返回类型无参数

基本语法

```
int main()
{
    //构建匿名函数 将构建的匿名函数存在lam函数指针下
    auto lam = [] {cout << "Lambda" << endl; };
    //调用
    lam();
    system("pause");
    return 0;
}
```

由于匿名函数无法直接操作，我们必须借助存储函数指针进行持有操作。

注意：这并不是给函数增加了调用的名称，只是构建了一个函数指针变量存储函数。当没有参数时可以省略括号。当没有返回类型时可以省略箭头

带有返回类型带有参数

基本语法

```
int main()
{
    //构建匿名函数 将构建的匿名函数存在lam函数指针下
    auto lam = [](int A, int B)->int { return A > B ? A : B; };
    //调用
    int Result = lam(50, 60);
    system("pause");
    return 0;
}
```

由于匿名函数无法直接操作，我们必须借助存储函数指针进行持有操作。

注意：返回类型放到箭头后，参数填入到括号中即可。返回类型也可以隐式提供。

```
int main()
{
    //构建匿名函数，但是没有明确指出返回类型，而是隐式推断返回类型
    auto lam = [](int A, int B) {return A > B ? A : B; };
    //调用函数
    int Result = lam(50, 60);
    system("pause");
    return 0;
}
```

对于匿名函数编写，我们需要注意语法格式。一般匿名函数会被当做参数构建用来传递逻辑段，例如截图中Fun函数需要一个函数指针参数，我们就可以构建匿名函数进行传入

```
void Fun(int(*pFun)(int))
{
    int Num = pFun(100);//调用函数
    cout << Num << endl;
}

int main()
{
    //调用函数，并将参数构建为匿名函数填入
    Fun([](int N)->int {return N * 10; });//打印1000
    system("pause");
    return 0;
}
```

编写即调用

在编写匿名函数时，可以在结束后加上括号，即表示执行到语句即调用函数，表达式如下

[capture](parameters)->return-type{body;}();

```
int main()
{
    //匿名函数
    [] {cout << "Hello" << endl; }();//加括号表示立即执行函数
    system("pause");
    return 0;
}
```

04 捕获参数（局部）

局部变量捕获概述

在匿名函数表达式中[capture](parameters)->return-type{body;}，Capture用来描述捕获外部变量访问方式说明符。对于外部访问参数，在**局部变量**访问时分为几个操作细节

- 只读捕获（=），内部只能读取外部参数，禁止修改
- 引用捕获（&），内部修改会影响到外部参数

捕获范围（作用域中）分为：局部捕获，全部捕获

对于全局变量只要存在有效访问即可使用，不在讨论范围

只读捕获（全部）

表达式格式[=](parameters)->return-type{body;}

用=符号表示，在调用域内的变量在匿名函数中均可使用用，但是只能读取，禁止操作。

```
int main()
{
    int A = 10;
    //构建匿名函数，参数捕获为只读捕获，去掉=内部访问A会出错
    [=]() { A = 100; };//由于时只读操作，所以禁止修改
    system("pause");
    return 0;
}
```

由于是全部捕获，所以在匿名函数中可以操作访问main函数中的所有变量

引用捕获（全部）

表达式格式[&](parameters)->return-type{body;}

用&符号表示，在调用域内的变量在匿名函数中均可使用（可读可写）。

```
int main()
{
    int A = 10;
    //构建匿名函数，参数捕获为引用捕获，去掉&内部访问A会出错
    auto lam = [&]() { A = 100; };//在匿名函数中访问外部变量A，并修改
    //调用函数
    lam();
    cout << A << endl;//输出100，因为是引用捕获，内部修改影响外部参数
    system("pause");
    return 0;
}
```

由于是全部捕获，所以在匿名函数中可以操作访问main函数中的所有变量

部分捕获

当存在多个变量时，可以采用部分捕获，并且有针对性的表明某些变量捕获的方式。

```
int main()
{
    int A = 0;
    int B = 0;
    int C = 0;
    [&] {};//可以访问到ABC并且可以修改
    [=] {};//可以访问到ABC，但是不能修改
    [A, &B] {};//可以访问到A（只读），B（可读可改），不能访问C
    [&, A] {};//可以访问到ABC，但是A是只读，其他可读可写
    [=, &A] {};//可以访问到ABC，但是只有A是可读可写，其他变量只读
    system("pause");
    return 0;
}
```

部分捕获动作只是针对希望操作的变量给与特殊描述，当存在多个可用逗号分开

05 捕获参数（成员）

成员参数捕获概述

当在类内部构建匿名函数时，在捕获参数中，局部变量捕获规则与第四章节一致。在捕获成员参数时，增加了一个捕获描述this，表达式[this](parameters)->return-type{body;}。

对于成员参数捕获，添加this，=，&（三者加任意均可访问成员内容）效果均一致，即在匿名函数中可以使用所有类成员变量（不受访问修饰符约束），**可以读取也可以修改。**

对于成员参数捕获，无进行部分捕获，也就是无法只针对某几个成员参数进行捕获（**容易造成与局部捕获冲突**）

切记混合捕获时的区别，大家可以理解为，只有有存在捕获标记，则成员参数均可修改和读取

```
class A
{
private:
    int N1 = 0;
public:
    void Fun()
    {
        int N = 0;
        [] {};//不可以捕获外部参数
        [this] {N1 = 100; N = 10; };//可读可修改 N只读
        [=] {N1 = 100; N = 10; };//N1可读可修改 N只读
        [&] {N1 = 100; N = 10; };//N1,N可读可修改
        [N1] {};//语法错误 当存在局部变量N1时，无法区分
        [&N1] {};//语法错误 当存在局部变量N1时，无法区分
        [this, N] {};//N1可读可写，N只读
        [this, &N] {};//N1,N可读可写
        [=, &N] {};//N1,N可读可写

    }
};
```

05 总结

- 匿名函数本身也是函数，具备函数的必须结构内容
- 匿名函数在编写结构时可以省略返回类型，通过隐式方式进行描述
- 当没有参数时，参数括号可以省略
- 匿名函数可以存储在函数指针变量中，用于操作
- 匿名函数在捕获参数时，可以进行外部内容捕获，通过设置捕获方式可以获取或是修改外部参数内容
- 在类内编写匿名函数时，对于类成员变量捕获，所有捕获方式均相同，即可以读取也可以修改成员参数内容

谢谢观看