華中科技大學

第12章 类型解析、转换与推导

Lambda表达式

许向阳

xuxy@hust.edu.cn

函数式程序设计

泛型程序设计

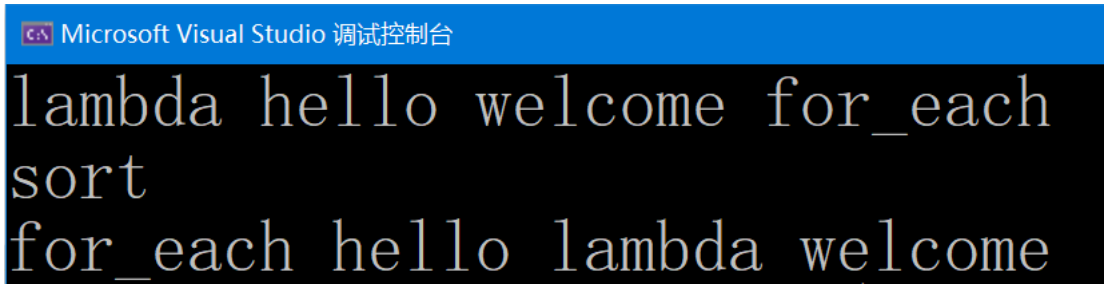
面向对象程序设计

神奇的Lambda表达式


```
#include <iostream>
#include <vector>
#include <algorithm>
using  namespace std;
int main()
{
    vector<int> vec = { 10,30,20,5,15,45 };
    for_each(vec.begin(), vec.end(), [](int x) {cout << x << " "; });

    cout << endl << "sort " << endl;
    sort(vec.begin(), vec.end(), [](int a, int b) {return a < b; });
    for_each(vec.begin(), vec.end(), [](int x) {cout << x << " "; });
}
```

神奇的Lambda表达式


```
#include <iostream>
#include <vector>
#include <algorithm>
using  namespace std;
int main()
{
    vector<string> vec = { "lambda","hello","welcome","for_each"};
    for_each(vec.begin(), vec.end(), [](string x) {cout << x << " "; });

    cout << endl << "sort " << endl;
    sort(vec.begin(), vec.end(), [](string a, string  b) {return a<b; });
    for_each(vec.begin(), vec.end(), [](string x) {cout << x << " "; });
}
```



神奇的Lambda表达式

// deepseek 编写的程序。题目如下
// 用 C++编程。设有类 Student, 包含 char name[10]; int age; int score.
// 定义 vector<Student> students,
// 用sort 对 students 中的元素排序，排序时，可以分别使用name、age，score。
// 输出排序后的结果。vector 中的元素自定。

程序参见：函数式编程_STL_Lambda示例.cpp

神奇的Lambda表达式


```
class Student {
public:      char name[10];      int age;      int score;
   Student(const char* n, int a, int s);
    void print() const ;
};

 vector<Student> students = {
        Student("Tom", 20, 85),
        Student("Alice", 22, 92),
        Student("Bob", 19, 78),
        Student("John", 21, 88),
        Student("Emma", 20, 95)
    };
```

神奇的Lambda表达式


```
sort(students.begin(), students.end(),
     [](const Student& a, const Student& b) {
         return strcmp(a.name, b.name) < 0;});
for (const auto& s : students)      // 按姓名排序
{     s.print();  }

sort(students.begin(), students.end(),
   [](const Student& a, const Student& b) {
        return a.age < b.age;});   按年龄排序

bool compareByScore(const Student& a, const Student& b)
{    return a.score < b.score; }        按分数排序
sort(students.begin(), students.end(), compareByScore);
```

神奇的Lambda表达式

对于简单的遍历，更简洁的表达形式

```
#include <iostream>
#include <vector>
#include <algorithm>
using  namespace std;
int main()
{
    vector<int> vec = { 10,30,20,5,15,45 };
    for (int x : vec) {cout << x << " "; };

    cout << endl << "sort " << endl;
    sort(vec.begin(), vec.end(), [](int a, int b) {return a > b; });
    for (int x : vec) { cout << x << " "; };
}
```

```
10 30 20 5 15 45
sort
45 30 20 15 10 5
```

神奇的Lambda表达式

for_each 是一个定义在 <algorithm> 头文件中的算法。
对序列中的每个元素[_First, _Last)执行一个函数 _Func。

```
template <class _InIt, class _Fn>
_CONSTEXPR20 _Fn for_each(_InIt _First, _InIt _Last,
_Fn _Func) {
_Adl_verify_range(_First, _Last);
    auto _UFirst       = _Get_unwrapped(_First);
    const auto _ULast = _Get_unwrapped(_Last);
    for (; _UFirst != _ULast; ++_UFirst) {
        _Func(*_UFirst);
    }
    return _Func;
}
```

函数模板　　以函数为参数

神奇的Lambda表达式

華中科技大學

```
void myfunc(int x)
{    x = 2 * x;
     cout << x << " ";
}

int main()
{
     vector<int> vec = { 10,30,20,5,15,45 };
     for_each(vec.begin(), vec.end(), myfunc);
}
```

```
20 60 40 10 30 90
```

```
for_each(_InIt _First, _InIt _Last, _Fn _Func)
```

迭代器取出的每个元素，都作为 _Func的参数

神奇的Lambda表达式


```
void myfunc(int x)
{    x = 2 * x;
     cout << x << " ";
}
int main()
{    vector<int> vec = { 10,30,20,5,15,45 };
     auto  mylambda = [](int x)
          {x = 2 * x;  cout << x << " "; };
     for_each(vec.begin(), vec.end(), myfunc);
     cout << endl;
     for_each(vec.begin(), vec.end(), mylambda);
}
```

```
20 60 40 10 30 90
20 60 40 10 30 90
```

for_each(vec.begin(), vec.end(), [](int x) {cout << x << " "; });

课堂目标

- Lambda表达式 的实现原理
  - 局部类、匿名类、（）函数重载
  - 定义Lambda表达式 =》
    - 定义匿名类， 类中的数据成员、函数成员、以及定义匿名类的对象
- Lambda 表达式的定义
  - 捕获列表 ： 捕获对象、捕获方式
  - 形参列表
  - 函数的返回类型、函数体、 异常说明
- Lambda 表达式的使用
  - 等同一个函数的用法　　**仿函数**

问题

```
void f()
{
    int  x=20,  y=30;
    char msg[20] = "good news";
        //  显示局部变量的值

    x = 25;  y = 35;  strcpy_s(msg, "good");
      //  第二次显示局部变量的值

    x = 1;   y = 2;   strcpy_s(msg, "bye bye");
      //  第三次显示局部变量的值
}
```

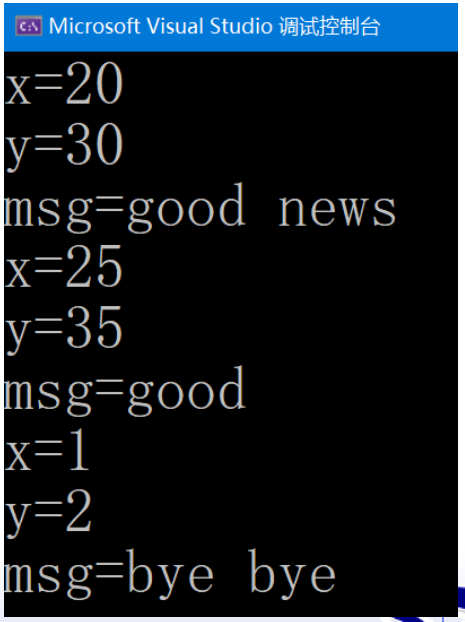
Q：怎样用一个函数完成局部变量的显示？

如果能在函数f中，再定义一个函数就好了！

不允许在函数中定义函数！

解决办法——局部类

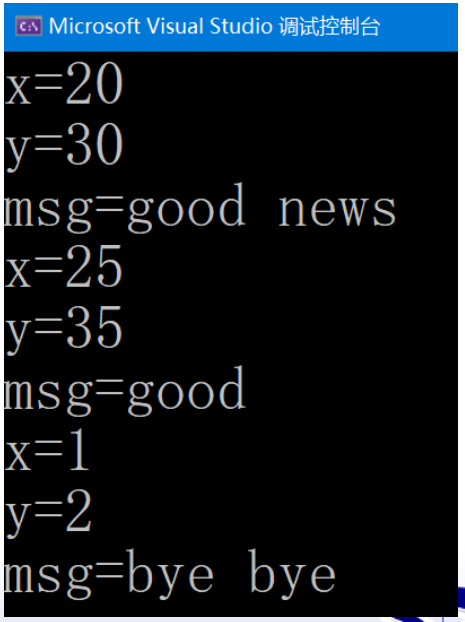
```
void f( ) {
    class info {          // 定义局部类
        private:  int  &n1,  &n2;    char *n3;
        public:
            info(int &u, int &v, char *p) :n1(u), n2(v), n3(p) {}
            void display() {
                cout << "x=" << n1 << endl;
                cout << "y=" << n2 << endl;
                cout << "msg=" << n3 << endl;
            }
    };
    int  x=20, y=30;
    char msg[20] = "good news";
    info a(x, y, msg);    // 对象创建……
    a.display( );  ……
}
```



解决办法之一 ——局部类

```
void f( ) {
    class info {    // 重载（）函数
        private:  int  &n1,  &n2;    char *n3;
        public:
            info(int &u, int &v, char *p) :n1(u), n2(v), n3(p) {}
            void operator( )( ) {
                cout << "x=" << n1 << endl;
                cout << "y=" << n2 << endl;
                cout << "msg=" << n3 << endl;
            }
    };
    int  x=20, y=30;
    char msg[20] = "good news";
    info a(x, y, msg);     ……
    a ( );  ……
}
```

使用 a( )， 就像在使用一个函数 f( );



解决办法之一 ——局部类


```
void main()
{    class info {
     public:
          void operator()(int x)
          { cout << x << " " ; }
     };
     vector<int> vec = { 10,30,20,5,15,45 };
     info a;
     for_each(vec.begin(), vec.end(), a);
}
```

```
10 30 20 5 15 45
```

- 局部类 **info;** 重载了 （）函数;
- 定义了局部类的对象 **a;**
- **For_each**中使用 **a,** 就像一个函数 **a(int x);**

解决办法——局部类

華中科技大學

讨论：在函数内部，实现一个“仿函数”

定义局部类、（）函数重载 的实现方法

有何优点？ 减少了全局符号的名字

解决函数泛滥、类泛滥等问题

有何缺点？ **定义局部类有些复杂**

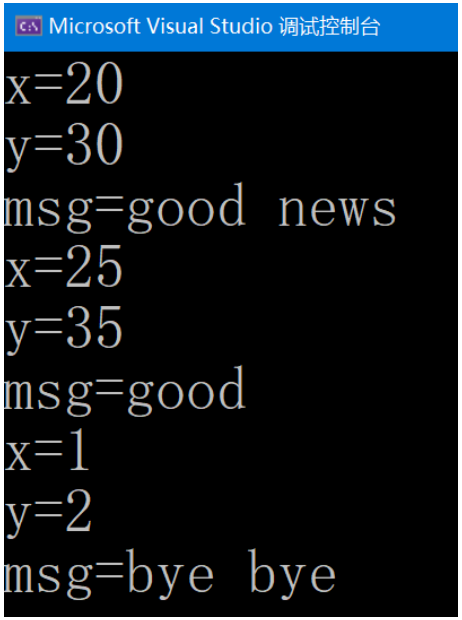
讨论：局部类、 嵌套类 有何差异？

嵌套类是一个独立的类，基本上与外围类无关

局部类只能在函数内部使用

解决局部类定义复杂的问题


```
void f()
{
    int  x = 20, y = 30;
    char msg[20] = "good news";

    auto mylambda = [&]( ) {
        cout << "x=" << x << endl;
        cout << "y=" << y << endl;
        cout << "msg =" << msg << endl;
    };
    mylambda( );
    x = 25;  y = 35;  strcpy_s(msg, "good");
    mylambda();
    x = 1;   y = 2;   strcpy_s(msg, "bye bye");
   mylambda();
}
```



解决局部类定义复杂的问题

字符数组的引用 [&]

```
class info {
    private: int &_x, &_y;     char (&_msg)[20];
    public:
    info(int& x, int& y, char (&msg)[20]) :
                    _x(x), _y(y), _msg(msg) {}
    void operator()() {
            cout << "x=" << _x << endl;
            cout << "y=" << _y << endl;
            cout << "msg=" << _msg << endl;
        }
    };

    int  x=20, y=30;
    char msg[20] = "good news";
    info a(x, y, msg);
```

解决局部类定义复杂的问题

```
void f()
{  auto mylambda = [&]( ) {
       cout << "x=" << x << endl;
       cout << "y=" << y << endl;
       cout << "msg =" << msg << endl;
   };
   mylambda( );
   ……
}
```

lambda表达式实现的基本原理

mylambda 是一个匿名类的**对象**；

匿名类的定义、对象构造，交给编译器去完成；

[有无代码自动生成的感觉？]

匿名类含有数据成员、构造函数、( )函数；

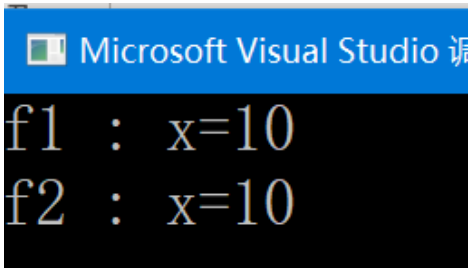
使用lambda表达式，本质是使用匿名类的（）函数

12.5 Lambda表达式

華中科技大學

带参数的 Lambda表达式

```
#include <iostream>
using namespace std;
void f1(int x)
{    cout << "f1 : x=" << x << endl; }

int main()
{   auto f2 = [](int x){cout << "f2 : x=" << x << endl;};
    f1(10);
    f2(10);
    return 0;
}
```

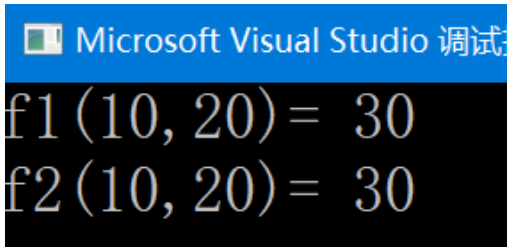


- f1是一个函数，是全局性的一个符号；
- f2是一个对象,是一个局部符号，只能在定义它的函数内部使用；
  f2 类似一个函数，匿名的函数;
- 在一个函数内部，是不能再定义函数的。

12.5 Lambda表达式

带参数的 Lambda表达式
带返回类型

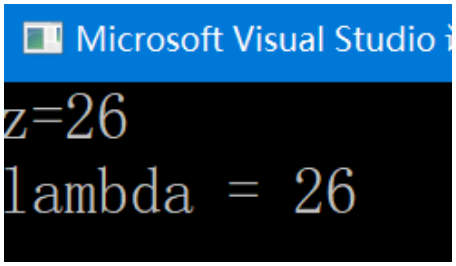
```
#include <iostream>
using namespace std;

int f1(int x,int y)
{    return x + y; }

int main()
{
    auto f2 = [](int x, int y) ->int {return x + y;};
    cout << "f1(10,20)= " << f1(10,20)<<endl;
    cout << "f2(10,20)= " << f2(10,20) <<endl;
    return 0;
}
```



- **Lambda 表达式 有返回类型**
- **用 lambda 表达式 取代了函数，表达更简洁**

12.5 Lambda表达式

➢ 有参数、有返回类型的圆括号函数




```
class A {
private: int x, y;
public:   A(int x,int y):x(x),y(y) { }
    int operator ()(int a, int b) { return x * y + a * b; }
};
int main()
{   int u = 2, v = 3;
    A   temp(u, v);
    int z = temp(4, 5);  // int z = temp.operator ()(4, 5);
    cout << "z=" << z << endl;
    la = [u, v](int a, int b)->int {return u * v + a * b;};
    cout << "lambda = " << la(4, 5) << endl;
    return 0;
}
```

Lambda 表达式中有捕获列表、有数据成员！

Q：会不会混淆对象定义的（）与（）函数？

12.5 Lambda表达式

```
z=26
lambda 1= 26
lambda 2 = 26
z=26
```

```
class A {
private: int x, y;
public:   A(int x,int y):x(x),y(y) { }
    int operator ()(int a, int b) { return x * y + a * b; }
};
int main()
{   int u = 2, v = 3;
    A   temp(u, v);
    int z = temp(4, 5);  // int z = temp.operator ()(4, 5);
    cout << "z=" << z << endl;
    auto la = [u, v](int a, int b)->int {return u * v + a * b;};
    cout << "lambda 1= " << la(4, 5) << endl;
    u = 10;  v = 20;
    cout << "lambda 2 = " << la(4, 5) << endl;
    z = temp(4, 5);
     cout << "z=" << z << endl;
    return 0;
}
```

Q：修改 u, v后，输出结果会不会变？

定义对象temp后，其数据成员不再变化

12.5 Lambda表达式

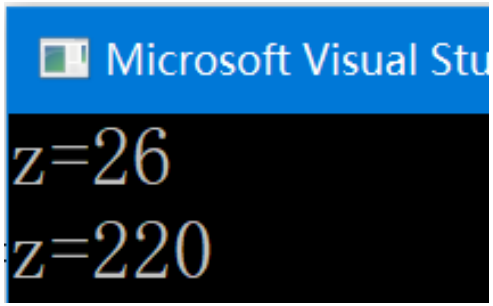
```
class A {
 private: int x, y;
 public:   A(int x,int y):x(x),y(y) { }
      int operator ()(int a, int b) { return x * y + a * b; }
};

int u,v ; ……
A   temp(u, v);
```

Q：怎样让A 的成员x, y随着 u, v 的变化，而变化？

A：将数据成员定义为引用！

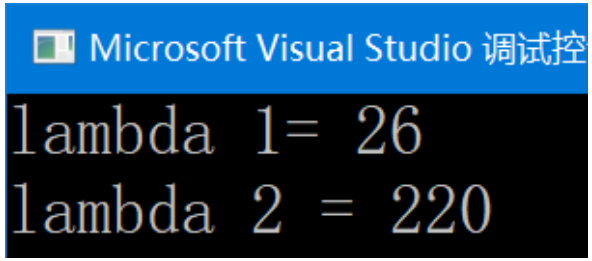
12.5 Lambda表达式

```
class A {
private: int &x, &y;
public:
    A(int &x, int &y) :x(x), y(y) { }
    int operator ()(int a, int b) { return x * y + a * b; }
};
void f1()
{    int u = 2, v = 3;
     A  temp(u, v);
     int z = temp(4, 5);
     cout << "z=" << z << endl;
     u = 10;    v = 20;
     z = temp(4, 5);
     cout << "z=" << z << endl;
}
```



12.5 Lambda表达式


```
void f()
{
    int u = 2, v = 3;
    auto la = [&u, &v](int a, int b)->int {return u * v + a * b;};
    cout << "lambda 1= " << la(4, 5) << endl;
    u = 10;  v = 20;
    cout << "lambda 2 = " << la(4, 5) << endl;
}
```



lambda 使用引用捕获成员，

匿名类的对象成为引用成员！

12.5 Lambda表达式

- Lambda表达式是C++引入的一种匿名函数、
  或者 匿名类的 （）函数
- Lambda表达式的声明格式

[捕获列表](形参列表)mutable 异常说明->返回类型{函数体}

[capture list](params list) mutable exception -> return type {

    function body }

capture list：捕获外部变量列表

mutable: 用于说明是否可以修改捕获变量

有mutable，则可以修改，但修改的是对象内的成员

exception: 异常接口说明

12.5 Lambda表达式

- **Lambda表达式的声明格式**

[捕获列表](形参列表)mutable 异常说明->返回类型{函数体}

[捕获列表](形参列表) ->返回类型{函数体}

[捕获列表](形参列表) {函数体}

[捕获列表]{函数体}

深入理解 Lambda表达式

➢ Lambda 表达式 实现的机理

```
zz = 0;
008420D0 C7 45 DC 00 00 00 00 mov         dword ptr [zz],0
auto surplus = [](int sal, int exp) ->int {return sal - exp;};
008420D7 33 C0                xor         eax,eax
008420D9 88 85 07 FF FF FF    mov         byte ptr [ebp-0F9h],al

    zz = surplus(100, 80);
008420DF 6A 50                push        50h
008420E1 6A 64                push        64h
008420E3 8D 4D D3             lea         ecx,[surplus]
008420E6 E8 85 F9 FF FF       call
<lambda_87ce62b9f68fc533a39a7753f0c263f7>::operator() (0841A70h)
008420EB 89 45 DC             mov         dword ptr [zz],eax
```

深入理解 Lambda表达式

mutable的理解

```
class lambda_xxxx {
private:
    int a;    int b;
public:
    lambda_xxxx(int _a,int _b) :a(_a),b(_b){ }
    int  operator( )(int x, int y) throw ()
    {        return  a + b + x + y ; }
};
```

如果lambda 表达式中，没有mutable，就相当于

int operator( )(int x, int y) const …

深入理解 Lambda表达式

准确理解 常成员函数

```
class lambda_yyyy {
private:
    int &x;        int y;
    char *p;
public:
    lambda_yyyy(int &_x,int _y) :x(_x){ …… }
    int  operator( )( ) const
    {
        y=20;    // 错误：不可修改 y
        x+=20;
        p = new char[10]; // 错误：不可修改 p
        *p =  'A' ;
        return 10;
    }
};
```

```
int & const x; int const y;
char * const p;
```

深入理解 Lambda表达式

- Lambda 表达式的本质就是重载了( )运算符的类
- 这种类通常被称为 functor, 即行为像函数的类
- lambda 表达式对象其实就是一个匿名的 functor
- 定义表达式时，就是定义一个类的对象
- 捕获列表中的变量，是对象构造函数的参数
- 在使用表达式时，捕获列表中的参数是不会传递的；

除非使用的是参数的引用，“间接”使用了参数的最新值。

12.5 Lambda表达式

- 捕获列表的参数用于捕获Lambda表达式的外部变量；
- 外部变量可以是函数参数或函数定义的局部自动变量；
- 出现“&变量名”表示引用外部变量；
- [&] 捕获“引用”所有函数参数或函数定义的局部自动变量。
- 出现“=变量名”表示使用外部变量的值（值参传递),
- [=]表示捕获所有函数参数或函数定义的局部自动变量的值；

混合使用隐式捕获和显式捕获

```
void  f4( ){
    int x = 10, y = 10, z = 10;
/*  当混合使用隐式捕获和显示捕获时，捕获列表的第一个元素必须是一个=或者&，指定隐式捕获方式；后面是显式捕获变量名列表，而且显式捕获的方式必须和隐式捕获不一样*/
    auto f3 = [=, &z]()-> int{ return x + y + z;};
        //隐式值捕获x , y，显式引用捕获变量z
    auto f4 = [&,x]()-> int{ return x + y + z;};
        //隐式引用捕获y , z，显式值捕获变量x}
}
```

12.5 Lambda表达式

- lambda可以直接使用它所在函数的局部static变量，以及在它所在函数之外声明的名字（例如全局的名字、所在文件的静态的名字），但这种情况不是捕获。
- lambda可以使用它所在函数的局部变量（包括它所在函数的形参），这个过程叫捕获。捕获只限于局部非static变量。
- 外部变量不能是全局变量或static定义的变量；
- 外部变量不能是类的成员；
- 参数表后有mutable表示在Lambda表达式可以修改“值参传递的值”，但不影响Lambda表达式外部变量的值。

12.5 Lambda表达式

```
void Lambdademo()
{    int  a = 1, b = 2,ret;
     auto lambda = [a, b](int x, int y) mutable ->
          int {  a+=5;     b += 10;
                 return a + b + x + y;   };
     ret = lambda(3, 4);
     cout<<"a= "<<a<<" b="<<b<<" ret = "<<ret<<endl;
     ret = lambda(3, 4);
     cout<<"a= "<<a<<" b="<<b<<" ret = "<<ret<<endl;
}
```

```
a= 1 b=2 ret = 25
a= 1 b=2 ret = 40
```

Q: 为什么会是该结果？

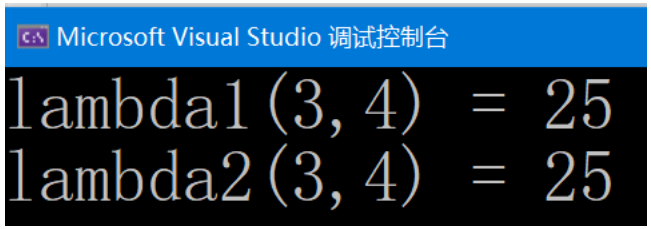
类中的内部数据成员修改与外部捕获变量无关

12.5 Lambda表达式

華中科技大學

```
void Lambdademo()
{
    int  a = 1, b = 2,ret;
    auto lambda = [a, b](int x, int y) ->int
        {  a+=5; //语法错误：表达式必须是可修改的左值
           b += 10; // 表达式必须是可修改的左值
           return a + b + x + y;
        };
    ret = lambda(3, 4);
    cout<<"a= "<<a<<" b="<<b<<" ret = "<<ret<<endl;
    ret = lambda(3, 4);
    cout<<"a= "<<a<<" b="<<b<<" ret = "<<ret<<endl;
}
```

12.5 Lambda表达式




```
void Lambdademo()
{
    int  a = 1, b = 2;
    auto lambda1 = [a, b](int x, int y) mutable ->
        int {  a += 5;    b += 10;
               return a + b + x + y;   };
    cout <<  "lambda1(3,4) = " <<lambda1(3,4) << endl;
    auto lambda2 = [a, b](int x, int y) mutable ->
        int {  a += 5;    b += 10;
               return a + b + x + y;   };
    cout << "lambda2(3,4) = " << lambda2(3,4) << endl;
}
```

lambda1 与 lambda2 无关，各自独立的数据成员

12.5 Lambda表达式

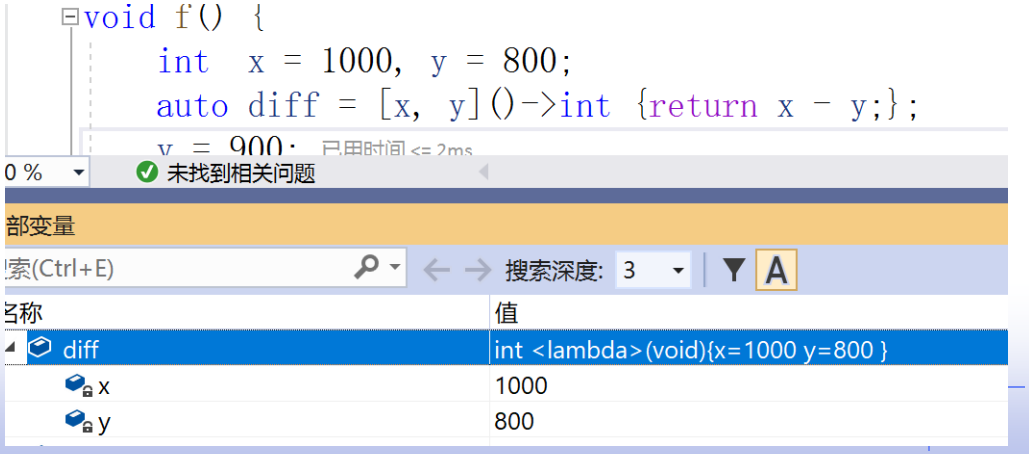
- Lambda表达式+捕获的自由变量叫闭包。
- 闭包(Closure)并不是一个新鲜的概念，很多函数式语言中都使用了闭包。
- 在JavaScript中，当在内嵌函数中使用外部函数作用域内的变量时，就是使用了闭包。
- 用一个常用的类比来解释闭包和类（Class）的关系：类是带函数的数据，闭包是带数据的函数。

测验

[捕获列表](形参列表)mutable 异常说明->返回类型{函数体}

```
void f( ) {
   int  x = 1000, y = 800;
   auto diff =  [x, y]( )->int {return x - y;};
   y = 900;
   cout << diff( ) << endl;  // 显示 200
   y = 700;
   cout << diff( ) << endl;
              // 显示 200
}
```

请根据原理解释看到的现象



测验

[捕获列表](形参列表)mutable 异常说明->返回类型{函数体}

```
void f2( ) {
    int  x = 1000, y = 800;
    auto diff =  [&]( )->int {return x - y;};
    y = 900;   // &捕获函数所有参数和函数内的局部自动变量
    cout << diff( ) << endl;    // 显示 100  :  1000 -900
    y = 700;
    cout << diff( ) << endl;    // 显示 300  :   1000-700
}
```

请根据原理解释看到的现象

测验

[捕获列表](形参列表)mutable 异常说明->返回类型{函数体}

```
void f3() {
    int  x = 1000, y = 800;
    auto diff = [ &x, y]()->int {x += 200; return x - y;};
    y = 900;            // 引用Lambda表达式的外部变量
    cout << diff() << endl;      // 400 :    1200 -800
    cout << x << endl;           //  1200
    y = 700;
    cout << diff() << endl;      //  600 :   1400 -800
}
```

请根据原理解释看到的现象

Lambda表达式应用示例


```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main( ) {
   auto f = [ ](int e)->void {cout << e << endl;};
    vector<int> v = { 1,2,3,4,5 };
    for_each(v.begin( ),  v.end( ),  f);  // 显示 1  2  3  4  5
     return 0;
}
```

Lambda表达式应用示例

華中科技大學

```
void myprint(int e) {
    cout << e << endl;
}
int main( ) {
   auto f = [ ](int e)->void {cout << e << endl;};
    vector<int> v = { 1,2,3,4,5 };
    for_each(v.begin( ),  v.end( ),  f);  // 显示 1  2  3  4  5
    for_each(v.begin(), v.end(), [](int e){ cout << e << endl;});
    for_each(v.begin(), v.end(), myprint);
     return 0;
}
```

总结

Lambda表达式的实现原理

Lambda表达式的定义和使用

auto 表达式名 = [捕获列表](形参列表)mutable 异常说明->返回类型{函数体};

[捕获列表](形参列表) {函数体}　//常见形式

函数式程序设计

Functional Programming

函数式编程

核心思想：强调通过纯函数的组合来构建程序；

- 函数是“第一等公民”（First-Class Citizens）
  - 函数与其他数据类型（如整数、字符串）地位平等；
  - 函数可以作为参数传递给其他函数、作为返回值返回，也可以赋值给变量或存储在数据结构中。
  - 这使得函数可以像数据一样被灵活操作，是高阶函数（接受或返回函数的函数）的基础。

函数式编程

强调纯函数（Pure Functions）

- 纯函数是指无副作用且结果仅由输入决定的函数；
- 无副作用：执行过程中不修改外部状态（如全局变量、输入参数、I/O 操作等），也不依赖外部状态的变化。
- 确定性：相同的输入始终产生相同的输出（类似数学中的函数概念，如 f(x) = 2x 对同一 x 结果唯一）。
- 纯函数避免了状态修改带来的不确定性，使程序更易推理、测试和并行化。

函数式编程

f(x) = x^2 + 3*x +5;

g(x) = 3*f(x) + x = 3*x^2 + 10*x +15;

定义输入数据和输出数据的关系

输入和输出间的一种映射（map）

函数不维护任何状态，核心精神是无状态(stateless)

不可变数据，输入数据不可变

函数式编程


```
int   copy_add(int x, int y)
{
        return x+y;
}
```

输入数据不可变

```
int   nocopy_add(int &x, int &y)
{
        x+=y;
       return x;
}
```

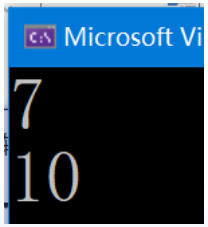
函数式编程


```
#include <iostream>
using namespace std;
int g(float (*f)(int), int a)
{    return  int((*f)(a)); }

float sqrt_minus_one(int x)
{     return float(sqrt(x) - 1);  }

float square_plus_one(int x)
{     return x * x + 1;  }

int main()
{   cout<<g(sqrt_minus_one, 64)<<endl;
     cout << g(square_plus_one, 3) << endl;
}
```

函数式编程

将一个函数作为参数，交给另一个函数

函数指针



函数式编程

華中科技大學

使用不可变对象

使用纯函数

使用高阶函数

使用 lambda 表达式

使用流 API

函数式编程


```
int  g(float (*f)(int), int a)
{
    return  int((*f)(a));
}

   auto p = [](int x) {return float(x * x + 1); };
    cout << g(square_plus_one, 3) << endl;
    cout << g(p, 4) << endl;
    cout << p(4) << endl;
    cout << g([](int x) {return float(x * x + 1); }, 5) << endl;
```

总结

- Lambda 表达式是函数式编程的重要工具；
- 它通过支持匿名函数和函数传递，让函数式编程的思想（如高阶函数、函数组合）更容易落地实现；
- 但它本身是一种语法特性，而非函数式编程的全部。
- 函数式编程的核心是“纯函数、不可变数据、声明式逻辑”等思想；
- Lambda 则是实现这些思想的常用手段之一。