lambda表达式：可调用对象

[capture list](parmeter list) -> return type {function type}

其中capture list(捕获列表)是一个lambda所在函数中定义的局部变量列表。

例：程序C++11\_test6

const int kSize = 8;

int size = 5;

static int size1 = 6;

*string* str("abcd");

// 捕获size

// 局部非static变量需要放入捕获列表中

// 全局变量和局部static变量就不需要了

auto l = [size](const *string*& a) {

return a.*size*() > size && a.*size*() > kSize && a.*size*() > size1;

};

l(str); // 调用lambda

值捕获和引用捕获

// 值捕获

int v1 = 42;

auto f1 = [v1]() {

// v1 = 5; // error，值捕获一般不能修改

return v1;

};

v1 = 0; // 不会影响lambda表达式f1通过值捕获获取的值

auto t1 = f1(); // t1 = 42

*cout* << "t1 = " << t1 << *endl*;

// 引用捕获

// 必须保证lambda在执行时捕获的变量存在

int v2 = 42;

auto f2 = [&v2]() {

return v2;

};

v2 = 0;

auto t2 = f2(); // t2 = 0;

*cout* << "t2 = " << t2 << *endl*;

// 隐式值捕获

// 不需要指明需要捕获的变量

auto f3 = [=]() {

return v1;

};

// 隐式引用捕获

auto f4 = [&]() {

return v2;

};

// 混合显示捕获和隐式捕获

// 隐式捕获和显示捕获的方式必须不同

// 隐式捕获必须在[]的第1位

// v2的捕获方式必须为引用捕获，

// 因为隐式捕获是值捕获

auto f5 = [=, &v2]() {

return v1 > v2;

};

// v2的捕获方式必须为值捕获

// 因为隐式捕获是引用捕获

auto f6 = [&, v2]() {

return v1 > v2;

};

可变lambda

例：程序C++11\_test7

// 可变lambda

// 值捕获的lambda一般不可以修改捕获的值

// 但可以通过添加关键字mutable来修改捕获的值

*size\_t* v1 = 42;

auto f1 = [v1]() mutable {return ++v1; };

v1 = 0;

auto j = f1(); // j = 43

指定lambda的返回类型：

例：程序C++11\_test7

// ->int显示指定lambda的返回类型

auto f3 = [](int i) ->int {

if (i > 0)

return i;

else

return -i;

};

auto k = f3(3);