STL预定义了一些function object和binder，后者允许合成更多精巧的function object。

STL预定义的function object

|  |  |
| --- | --- |
| 表达式 | 效果 |
| negate<type>() | -param |
| plus<type>() | param1 + param2 |
| minus<type>() | param1 - param2 |
| multiplies<type>() | param1 \* param2 |
| divides<type>() | param1 / param2 |
| modulus<type>() | param1 % param2 |
| equal\_to<type>() | param1 == param2 |
| not\_equal\_to<type>() | param1 != param2 |
| less<type>() | param1 < param2 |
| greater<type>() | param1 > param2 |
| less\_equal<type>() | param1 <= param2 |
| greater\_equal<type>() | param1 >= param2 |
| logical\_not<type>() | !param |
| logical\_and<type>() | param1 && param2 |
| logical\_or<type>() | param1 || param2 |
| bit\_and<type>() | param1 & param2 |
| bit\_or<type>() | param1 | param2 |
| bit\_xor<type>() | param1 ^ param2 |

对象排序或比较（排序函数和关联容器），默认是less<>，unordered容器默认为equal\_to<>。

function adapter（函数适配器）：指能够将不同的function object（或是和某值或某一般函数）结合起来的东西，本身也是一个function object。

C++11提供的function adapter

|  |  |
| --- | --- |
| 表达式 | 效果 |
| bind(op, args…) | 将args绑定给op |
| mem\_fn(op) | 调用op()，将它当作某个object（或pointer to object）的成员函数 |
| not1(op) | 单参反相：!op(param) |
| not2(op) | 双参反相：!op(param1, param2) |

bind()：最重要的adapter。

bind()的基本功能：将参数绑定于callable object，因此，如果一个函数、成员函数、function object或lambda需要若干参数，你可以将参数绑定为“明确指出的”或“被传入的”实参。明确指出实参，需写出其名称；使用被传入的实参，则可利用预定义占位符\_1, \_2,..。

例：程序stl\_test59

auto plus10 = *bind*(*plus*<int>(), *\_1*, 10);

*cout* << "+10: " << plus10(7) << *endl*;

// 组合两个STL预定义的functon object

auto plus10times2 = *bind*(*multiplies*<int>(), *bind*(*plus*<int>(), *\_1*, 10), 2);

*cout* << "(+10) \* 2: " << plus10times2(7) << *endl*;

auto pow3 = *bind*(*multiplies*<int>(), *bind*(*multiplies*<int>(), *\_1*, *\_1*), *\_1*);

*cout* << "x\*x\*x: " << pow3(7) << *endl*;

auto inversDivide = *bind*(*divides*<double>(), *\_2*, *\_1*);

*cout* << "invdiv: " << inversDivide(49, 7) << *endl*;

输出为：

+10: 17

(+10) \* 2: 34

x\*x\*x: 343

invdiv: 0.142857

bind()调用全局函数

例：程序stl\_test59

char MyToupper(char c)

{

*locale* loc;

return *use\_facet*<*ctype*<char>>(loc).*toupper*(c);

}

// 调用全局函数

*string* s("Internationalization");

*string* sub("Nation");

*string*::*iterator* pos

= *search*(s.*begin*(), s.*end*(), // string to search in

sub.*begin*(), sub.*end*(), // substring to search

*bind*(*equal\_to*<char>(), *bind*(MyToupper, *\_1*), // 比较标准

*bind*(MyToupper, *\_2*)));

if (pos != s.*end*())

{

*cout* << "\"" << sub << "\" is part of \"" << s << "\"" << *endl*;

}

输出为：

"Nation" is part of "Internationalization"

bind()调用成员函数：

例：程序stl\_test60

class Person {

public:

Person(const *string*& name) : name\_(name) {

}

void Print() const {

*cout* << name\_ << *endl*;

}

// 不能使用函数重载

// bind()无法判断绑定哪个函数

void Print2(const *string*& prefix) const {

*cout* << prefix << name\_ << *endl*;

}

private:

*string* name\_;

};

int main(int argc, char\* argv[])

{

*vector*<Person> coll = {Person("Tick"), Person("Trick"), Person("Track")};

// bind(&Person::Print, \_1)第一个实参为成员函数

// 下一个实参定义“用以调用成员函数”的对象

// 其他任何实参都会被传递给该成员函数

*for\_each*(coll.*begin*(), coll.*end*(), *bind*(&Person::Print, *\_1*));

*cout* << *endl*;

*for\_each*(coll.*begin*(), coll.*end*(), *bind*(&Person::Print2, *\_1*, "Person: "));

*cout* << *endl*;

// 单独使用bind

*bind*(&Person::Print2, *\_1*, "This is: ")(Person("nico"));

Person\* ptr = new Person("Jam");

// 传递对象的指针给bind

*bind*(&Person::Print2, *\_1*, "This is: ")(ptr);

delete ptr;

// 传递smart pointer给bind

*shared\_ptr*<Person> sp(new Person("Tom"));

*bind*(&Person::Print2, *\_1*, "This is: ")(sp);

return 0;

}

输出：

Tick

Trick

Track

Person: Tick

Person: Trick

Person: Track

This is: nico

This is: Jam

This is: Tom