10 泛型算法

1，概述

编程：使用find实现在vector中查找特定值

2，初识泛型算法

标准库提供了超过100个算法，这些算法有一致的结构。

只读算法：accumulate，equal

写容器元素的算法：fill, fill\_n, back\_inserter, copy, replace

重排容器元素的算法：sort

3，定制操作

除了函数和函数指针这两种可调用对象，还有两种可调用对象：重载了函数调用运算符的类和lambda表达式。

lambda表达式的形式：[capture list](parameter list) -> return type{function body}

类似参数传递，变量的捕获方式也可以是值或引用。

我们可以从一个函数返回lambda，那么此lambda也不能包含引用捕获。

隐式捕获：可以让编译器根据lambda体中的代码来推断我们使用哪些变量，&告诉编译器采用捕获引用方式，=表示采用值捕获方式。

例子：[=](const string &s){return s.size() >= sz;}

对一部分变量采用值捕获，对其他变量采用引用捕获，可以混合使用隐式捕获和显示捕获。

例子：[&,c](const string &s){os << s << c;}和[=, &os](const string &s){os << s << c;}

如果只有一个return语句可以让编译器推断返回值类型，反之需要自己指定返回值类型。

[](int i) -> int {if(i < 0) return -i; else return i;}

如果lambda的捕获列表为空，通常可以使用函数来代替它，对于捕获局部变量的lambda，用函数替换它不是那么容易。

对于以上问题我们使用bind标准库函数，它可以看作是一个通用的函数适配器。

bind的一般形式为：auto newCallable = bind(callable, arg\_list);

bind(check\_size, \_1, sz)调用check\_size的第二个参数传给sz。

名字\_n都定义在一个名为placeholders的命名空间中。需要首先声明 using std::placeholders::\_1;更简单的方法是using namespace std::placeholders;

auto g = bind(f, a, b, \_2, c, \_1);调用g( \_1, \_2)映射为f( a, b, \_2, c, \_1)

与lambda类似，有时对有些绑定的参数我们希望以引用方式传递。使用ref函数。

bind(print, ref(os), \_1, ' ')

4 再探迭代器

除了为每个容器定义迭代器之外，标准库在头文件iterator中还定义了几种迭代器：插入迭代器，流迭代器，反向迭代器和移动迭代器。

5 泛型算法结构

6 特定容器算法

链表类型list和forward\_list定义了几个成员函数形式的算法，定义了独有的sort,merge,remove,reverse和unique。

通用版本的sort要求随机访问迭代器，因此不能用于list和forward\_list,因为这两个类型分别提供双向迭代器和前向迭代器。

参考文档：list和forward\_list成员函数版本的算法