**10 泛型算法**

**1，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，sort,find等。

find( vec.cbegin(), vec.cend(), val)

**2，lambda**

四种可调用对象：函数，函数指针，重载了运算符的类和lambda表达式。

一个lambda表达式的形式：[capture list](parameter list) -> return type{function body}

例子：[](const string &a, const string &b){return a.size() < b.size()}

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不能包含引用捕获。

对一个值被拷贝的变量，lambda不会改变其值，如果我们希望改变变量的值，使用关键字mutable，或者通过捕获引用。

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

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;}

**3，参数绑定bind**

如果lambda的捕获列表为空，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placeholders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, ' ')