C++标准程序库读书笔记

仿函数（函数对象）

仿函数的定义方法

```
class FunctionObjectType
{
    public:
    void operator()()
    {
        statements
    }
};
```

这种定义方式的好处：

- 仿函数比一般函数更灵巧，因为它可以拥有状态。对于仿函数，可以同时拥有两个状态不同的实体，一般函数则不能这样。
- 每个仿函数都有其型别，因此你可以将仿函数的型别当作template参数来传递，从而指定某种行为模式，此处还有一个好处，容器型别也会因为仿函数的不同而不同。
- 执行速度上，仿函数通常比函数指针更快。

仿函数是传值，不是传址。因此算法并不会改变岁参数而来的仿函数的标志。

将仿函数以传值方式传递的优点和缺点：

- 优点：可以传递常量和暂时表达式。
- 缺点：无法改变仿函数的状态，算法可以改变仿函数的状态，但你无法存取并改变最终状态，因为改变的只不过是仿函数的副本。

有两个办法可以从运用了仿函数的算法中获取结果或反馈：

1. 以by reference的方式传递仿函数
2. 运用for_each()算法的回返值。

判断式和仿函数

判断式：返回布尔值（可转换为bool）的一个函数或仿函数。对STL而言，并非所有返回布尔值的函数都是合法的判断式。

预定义的仿函数

仿函数	效果
negate()	-param
plus()	param1+param2
minus()	param1-param2
multiplies()	param1*param2
divides()	param1/param2
modulus()	param1%param2
equal_to()	param1==param2
not_equal_to()	param1!=param2
less()	param1<param2
greater()	param1>param2
less_equal()	param1<=param2
greater_equal()	param1>=param2
logical_not()	!param
logical_and()	param1&&param2
logical_or()	param1

函数配接器

函数配接器是指能够将仿函数和另一个仿函数结合起来的仿函数。声明于中。

表达式	效果
bind1st(op,value)	op(value,param)
bind2nd(op,value)	op(value,param)
not1(op)	!op(param)
not2(op)	!op(param1,param2)

为成员函数预定义的仿函数

表达式	效果
mem_fun——ref(op)	调用op,那是某对象的一个const成员函数
mem_fun(op)	调用op,那是某对象指针的一个const成员函数

被mem_fun_ref和mem_fun两者都以无参数或单参数方式来调用成员函数。

针对一般函数而设计的函数配接器

表达式	效果
ptr_fun(op)	op(param) op(param1,param2)

辅助用仿函数

一般而言，所有的函数行文都可以藉由仿函数的组合而实现。

下面这些组合型配接器很有用

- f(g(elem))

  > 这是一元组合函数的一般形式，一元判断式被嵌套调用，g()执行结果被当作是f()参数，整个表达式的操作类似一个一元判断式。

- f(g(elem1,elem2))

  > 这个形式中，两元素elem1和elem2作为参数传给二元判断式g(),其结果在作为参数传给一元判断式f(),整个表达式的操作类似一个二元判断式。

- f(g(elem),h(elem))

  > 此处，elem作为参数被传递给两个不同的一元判断式g()和h(),两者的结果由二元判断式f()处理，这种形式系以某种办法将单一参数注射到一个组合函数中，整个表达式的操作类似一个一元判断式。

- f(g(elem1),h(elem2))

  > 此处elem1和elem2分别作为唯一参数传给两个不同的一元判断式g()和h(),两个结果共同被二元判断式f()处理，某种程度上这种形式系在两个参数身上分布一个组合函数，整个表达式的操作类似一个二元判断式。

后面还有一部分知识有待完善