C++语言程序设计

贺利坚 主讲

STL基本算法

STL算法基础

- ▶ STL算法本身是一种函数模版
  - □ 通过迭代器获得输入数据
  - □ 通过函数对象对数据进行处理
  - □ 通过迭代器将结果输出
- ▶ STL算法是通用的，独立于具体的数据类型、容器类型
- ▶ STL算法分类
  - □ 不可变序列算法
  - □ 可变序列算法
  - □ 排序和搜索算法
  - □ 数值算法

```
template < class InputIterator, class OutputIterator, class UnaryOperator >
OutputIterator transform (InputIterator first1,  InputIterator last1,
    OutputIterator result,  UnaryOperator op );
```

```
template<class RandomAcessIterator>
void sort (RandomAcessIteratorr first,RandomAcessIterator last)
```

```
template<class InputIterator, class OutputIterator,>
void copy(InputIterator first, InputIterator last, Type val,
                OutputIterator result);
```

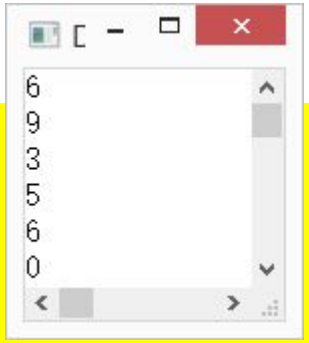
```
template<class InputIterator, class Type, class BinaryFunction>
Type accumulate(InputIterator first,  InputIterator last, Type val,
                        BinaryFunction binaryOp);
```

```
template<class InputIterator, class UnaryPredicate>
InputIterator find_if ( InputIterator first,  InputIterator last,
                                UnaryPredicate pred )
```

不可变序列算法

- 不可变序列算法
  - 不直接修改所操作的容器内容的算法
  - 用于查找指定元素、比较两个序列是否相等、对元素进行计数等
- 包括
  - for_each()：将一个非修改式函数用于指定区间中的每个成员。
  - find()：在区间中查找某个值首次出现的位置。
  - find_if()：在区间中查找第一个满足断言测试条件的值。
  - find_end()：在序列中查找最后一个与另一个序列匹配的值。匹配时可使用等式或二元断言。
  - find_first_of()：在第二个序列中查找第一个与第一个序列的值匹配的元素。
  - adjacent_find()：查找第一个与其后面元素匹配的元素。匹配时可使用等式或二元断言。
  - count()：返回特定值在区间中出现的次数。
  - count_if()：利用输入的条件，对标志范围内的元素进行操作，返回结果为true的个数。
  - mismatch()：比较两个序列，找出首个不匹配元素的位置。
  - equal()：比较两个序列元素是否相同。
  - search()：在一个系列中查找另一个系列第一次出现的位置。
  - search_n()：在指定范围内查找某值val出现n次的子序列。

不可变序列算法示例




```
#include <iostream>
#include <algorithm>
#include <functional>
#include <vector>
using namespace std;
```

```
int main()
{
    int iarray[] = { 0, 1, 2, 3, 4, 5, 6, 6, 6, 7, 8 };
    vector<int> ivector(iarray, iarray + sizeof(iarray) / sizeof(int));
    int iarray1[] = { 6, 6 };
    vector<int> ivector1(iarray1, iarray1 + sizeof(iarray1) / sizeof(int));
    int iarray2[] = { 5, 6 };
    vector<int> ivector2(iarray2, iarray2 + sizeof(iarray2) / sizeof(int));
    int iarray3[] = { 0, 1, 2, 3, 4, 5, 7, 7, 7, 9, 7 };
    vector<int> ivector3(iarray3, iarray3 + sizeof(iarray3) / sizeof(int));
    cout << *adjacent_find(ivector.begin(), ivector.end()) << endl; //找出ivector之中相邻元素值相等的第一个元素
    cout << count_if(ivector.begin(), ivector.end(), bind2nd(less<int>(), 7)) << endl; //找出ivector之中小于7的元素个数
    cout <<  *find_if(ivector.begin(), ivector.end(), bind2nd(greater<int>(), 2)) << endl; //找出ivector中大于2的第一个元素
    cout << *search(ivector.begin(), ivector.end(),ivector2.begin(), ivector2.end()) << endl; //子序列ivector2在ivector中出现的起点位置元素
    cout << *search_n(ivector.begin(), ivector.end(), 3, 6, equal_to<int>()) << endl;//查找连续出现3个6的起点位置元素
    cout << equal(ivector.begin(), ivector.end(), ivector3.begin()) << endl; //判断两个区间ivector和ivector3相等否
    return 0;
}
```

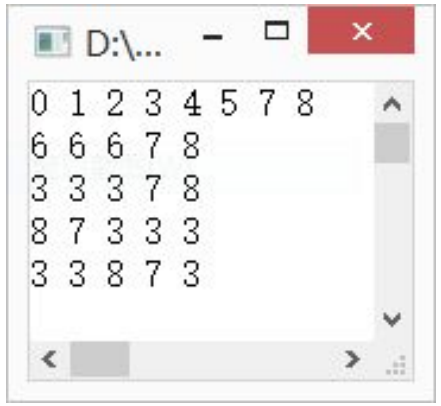
可变序列算法：可以修改它们所操作的容器对象

▶包括对序列进行复制、删除、替换、倒序、旋转、交换、变换、分割、去重、填充、洗牌的算法及生成一个序列的算法

- copy：复制序列
- copy_backward：元素以相反顺序被复制
- iter_swap：交换两个ForwardIterator的值
- remove：删除指定范围内所有等于指定元素的元素
- remove_copy：将所有不匹配元素复制到一个制定容器
- remove_if：删除指定范围内输入操作结果为true的所有元素
- replace：将指定范围内所有等于vold的元素都用vnew代替。
- replace_copy：与replace类似，不过将结果写入另一个容器。
- replace_if：将指定范围内所有操作结果为true的元素用新值代替。
- swap：交换存储在两个对象中的值。
- swap_range：将指定范围内的元素与另一个序列元素值进行交换。
- unique：清除序列中重复元素
- unique_copy：与unique类似，不过把结果输出到另一个容器。

可变序列算法示例

```
#include <iostream>
#include <algorithm>
#include <functional>
#include <iterator>
#include <vector>
using namespace std;
```




```
int main()  //略去了每个操作之后的输出
{
    int iarray1[]= { 0, 1, 2, 3, 4, 4, 5, 5, 6, 6, 6, 6, 6, 7, 8 };
    int iarray2[] = { 0, 1, 2, 3, 4, 5, 6, 6, 6, 7, 8 };
    vector<int> ivector1(iarray1, iarray1 + sizeof(iarray1) / sizeof(int));
    vector<int> ivector2(iarray2, iarray2 + sizeof(iarray2) / sizeof(int));
    ostream_iterator<int> output(cout, " ");
    vector<int> ivector4;
    //将删除元素6后的ivector2序列置于另一个容器ivector4之中
    remove_copy(ivector2.begin(), ivector2.end(), back_inserter(ivector4), 6);
    //删除小于6的元素: remove_if实现逻辑删除，erase实施真正的物理删除
    ivector2.erase(remove_if(ivector2.begin(), ivector2.end(), bind2nd(less<int>(), 6)), ivector2.end());
    //将所有的元素值6，改为元素值3
    replace(ivector2.begin(), ivector2.end(), 6, 3);
    //逆向重排每一个元素
    reverse(ivector2.begin(), ivector2.end());
    //旋转（互换元素）：将[first, middle), 和[middle, end)中的元素互换并输出结果
    rotate_copy(ivector2.begin(), ivector2.begin() + 3, ivector2.end(), output);
    return 0;
}
```

可变序列算法示例(续)

```
#include <iostream>
#include <algorithm>
#include <functional>
#include <iterator>
#include <vector>
using namespace std;

class evenByTwo
{
private:
    int x;
public:
    evenByTwo(): x(0) { }
    int operator () ()
    {
        return x += 2;
    }
};
```

```
int main()
{
    vector<int> ivector3(4);
    ostream_iterator<int> output(cout, " ");

    //迭代遍历ivector3区间，每个元素填上-1
    fill(ivector3.begin(), ivector3.end(), -1);
    copy(ivector3.begin(), ivector3.end(), output);  // 使用copy进行输出
    cout << endl;

    //迭代遍历ivector3区间，对每一个元素进行evenByTwo操作
    generate(ivector3.begin(), ivector3.end(), evenByTwo());
    copy(ivector3.begin(), ivector3.end(), output);
    cout << endl;

    return 0;
}
```