

# 第5章 使用序列式容器 并分析字符串

刘 卉

huiliu@fudan.edu.cn



# 前言

## □ 深入理解如何使用标准库

- 一致性体系结构: 学会使用一种容器→使用所有库容器.  
e.g. 一个string对象当作一个vector使用.
- 所有库类型在操作逻辑上相同.
- 标准库的构造使得不同类型的等价操作以相同方式工作.

# 主要内容

5.1 把学生分类

5.2 迭代器(iterator)

5.3 用迭代器取代索引

5.4 重新设计数据结构以获取更好性能

5.5 list类型

5.6 剖析string类

小结



# 5.1 把学生分类

## □ 两类: 通过 & 未通过

- 检测每条学生记录: 通过的记录保存在一个vector对象中, 未通过的记录保存在另一个vector对象中.

- 功能函数fgrade:

```
// predicate to determine whether a student failed
```

```
bool fgrade(const Student_info& s)
```

```
{
```

```
    return grade(s) < 60;
```

```
}
```

```
// separate passing and failing student records: first try
vector<Student_info> extract_fails(vector<Student_info>& students)
{
    vector<Student_info> pass, fail;
    for (vector<Student_info>::size_type i = 0;
         i != students.size(); ++i)
        if (fgrade(students[i]))
            fail.push_back(students[i]);
        else
            pass.push_back(students[i]);
    students = pass; // 修改实参为'通过的学生记录'
    return fail;    // 返回'未通过的学生记录'
}
```

未通过的

通过的



## 5.1.1 就地删除元素

### 潜在缺点

- 需要足够内存以保存每个学生的两份副本.

### 解决方法

- 从students中删除未通过学生;
- 只创建fail保存未通过学生.

```
// second try: correct but potentially slow
vector<Student_info> extract_fails(vector<Student_info>& students)
{
    vector<Student_info> fail;
    vector<Student_info>::size_type i = 0;
    // invariant: elements [0, i) of students represent passing grades
    while (i != students.size()) {
        if (fgrade(students[i])) {
            fail.push_back(students[i]);
            students.erase(students.begin() + i);
        } else {
            ++i;
        }
    }
    return fail;
}
```

why?

不能直接对索引操作

思考: 用for循环行吗? 为什么?  
for (i = 0; i != students.size(); ++i)



# erase函数

所有容器都提供erase成员函数, 但不是所有容器都支持索引.

一致性原则: 提供相同工作方式的erase函数.

erase函数会改变vector对象的长度并调整索引.

为什么在循环条件中每次都要调用students.size()?





# 从vector中删除元素非常慢

vector针对快速随机访问作了优化.

代价: 插入/删除不在末尾的元素时, 开销较大.

大规模数据不应采用这种方法.

解决方法: 使用更合适的数据结构/更好的算法.



## 5.1.2 顺序访问和随机访问

### 随机访问

- 按任意顺序访问容器中的元素.
- 索引提供随机访问的能力.

### extract\_fails函数只需顺序访问容器

- 不需要索引.
- 重写函数, 仅使用支持顺序访问的操作来遍历各元素.
- 迭代器: C++标准库提供的类型.



## 5.2 迭代器(iterator)

迭代器是一个值, 它能够:

- 标识一个容器和其中的元素
- 检测元素的值
- 提供在元素之间移动的操作
- 约束可用操作



# 把使用索引的程序用迭代器重写

```
for (vector<Student_info>::size_type i = 0; i != students.size(); ++i)
    cout << students[i].name << endl;
```

```
for (vector<Student_info>::const_iterator iter = students.begin();
     iter != students.end(); ++iter)
    cout << (*iter).name << endl;
```

初始指向容器首元素



# 两种迭代器类型

## □ 每种标准库容器都有如下类型定义

container-type::const\_iterator

container-type::iterator

自动类型转换: iterator  $\rightarrow$  const\_iterator

```
vector<Student_info>::const_iterator iter = students.begin();
```

类型成员, 对容器对象进行只读访问

返回iterator类型值



# 迭代器操作

```
for (vector<Student_info>::const_iterator iter = students.begin();  
    iter != students.end(); ++iter)  
    cout << (*iter).name << endl;    // (*iter).name  $\leftrightarrow$  iter->name
```

- 可比较两个迭代器是否相等: ==, !=
- ++iter: 为迭代器重载的自增操作符, 使迭代器指向容器中下一个元素;
- 通过迭代器和引用操作符\*(或->)访问所指元素, (\*iter)是一个左值.



## `students.erase(students.begin()+i)`的含义

- ❑ `students.begin()`返回指向`students`首元素的迭代器, 加`i`指向第`i+1`个元素.
- ❑ 如果`students`是一个不支持根据索引随机访问的容器,
  - 可通过迭代器顺序访问, 但无法随机访问.
  - `students.begin()`返回的迭代器没有定义`'+'`操作→`students.begin()+i`编译无法通过.



## 5.3 用迭代器取代索引

```
// version 3: iterators but no indexing; still potentially slow
vector<Student_info> extract_fails(vector<Student_info>& students)
{
    vector<Student_info> fail;
    vector<Student_info>::iterator iter = students.begin();
    while (iter != students.end()) {
        if (fgrade(*iter)) {
            fail.push_back(*iter);
            iter = students.erase(iter);
        } else
            ++iter;
    }
    return fail;
}
```

不再是const\_iterator

每次循环均调用

使iter无效, 返回被删除元素之后元素的迭代器





## 5.4 重新设计数据结构以获取更好性能

对于小型输入，  
用vector实现的  
程序性能良好，  
但随着输入增加，  
性能急剧下降。

- 在尾部插入/删除元素，性能良好。
- 在中间插入/删除元素时，该元素之后的所有元素都要移动。



# 重新设计数据结构以获取更好性能

需要一种数据结构, 能高效地插入/删除容器中任何一个元素.

- 无需支持索引.
- 只需支持迭代器.



## 5.5 list类型

list也是一种容器类型

- 支持在容器中的任意位置进行快速插入/删除.
- 结构较复杂, 顺序访问比vector慢.
- list与vector有很多相同操作, 但list不支持索引(即, 不支持随机访问).

// version 4: use *list* instead of *vector*

```
list<Student_info> extract_fails( list<Student_info>& students)
{
    list<Student_info> fail;
    list<Student_info>::iterator iter = students.begin();
    while (iter != students.end()) {
        if (fgrade(*iter)) {
            fail.push_back(*iter);
            iter = students.erase(iter);
        } else
            ++iter;
    }
    return fail;
}
```

## 5.5.1 重要区别

### 插入/删除操作对vector迭代器的影响

- erase操作: 被删除元素及其之后元素的迭代器均失效.
- push\_back操作: 所有迭代器失效. 为新元素分配空间可能会引起整个vector的重新分配.
- 在循环中使用插入/删除操作时, 必须确保没有保存任何无效的迭代器.

### 插入/删除操作对list迭代器的影响

- erase/push\_back不会使指向其它元素的迭代器失效.
- 指向被删除元素的迭代器失效, 因为该元素已不存在.

## □ list类不能使用标准库的sort函数

- 提供自己的sort成员函数

```
list<Student_info> students;  
students.sort(compare);
```

```
vector<Student_info> students;  
sort(students.begin(), students.end(), compare);
```



## 5.5.2 为什么要如此麻烦?

### □ 不同数据结构对性能的影响

- 只需随机访问→选择vector
- 需从容器中删除/向容器中添加元素→选择list

File size	list	vector
735	0.1	0.1
7,350	0.8	6.7
73,500	8.8	597.1



## 5.6 剖析string类

### □ string可看作特殊的容器

- 支持部分容器操作: 索引, 提供迭代器.
- 很多用于vector的技术都可用于string.

e.g. `string s;`

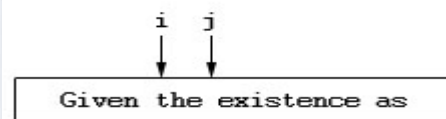
`s[0]`: 第一个字符,

`s[s.size()-1]`: 最后一个字符.



## □ 问题描述: 把一行文本分成多个单词

- 写成函数: 带一个字符串参数, 返回`vector<string>`类型值, 包含由空白符分隔的所有单词.
- 定义索引`i`和`j`, 通过计算`i`和`j`的值给每个单词定位`[i, j)`.



## □ string的成员函数substr

`string` `string::substr( size_type i, size_type n) const;`

- 从位置`i`开始复制`n`个字符以创建一个新字符串.

## □ <cctype>头文件

- 定义了isspace等处理单个字符的函数.
- cctype库: C++从C继承而来.





# 小结

## □ 容器和迭代器

- 在标准库的设计中, 不同容器的相似操作具有相同接口和语义.  
e.g. vector和list都支持push\_back(\*iter)和erase(iter)操作.
- 支持随机访问的容器和string类型可用c[n]取元素.

■ 所有序列式容器和string类型都提供如下操作:

`container<T> c`      `container<T> c(c2)`      `container<T> c(n)`

`container<T> c(n, t)`      `container<T> c(b, e)`

`container<T> c = c2`      `c = c2`

`container<T>::iterator`      `container<T>::const_iterator`

`container<T>::size_type`

`c.size()`      `c.empty()`      `c.push_back(t)`

`c.begin()`      `c.end()`      `c.rbegin()`      `c.rend()`

`c.erase(it)`      `c.erase(b, e)`

`c.insert(it, t)`      `c.insert(it, b, e)`

## □ 迭代器操作

`*it, (*it).x, it->x, ++it, it++, b == e, b != e`

- string类型提供迭代器, 支持随机访问

`s.substr(i, j) getline(is, s) s += s2`

- vector提供库容器中功能最强大的迭代器——随机访问迭代器

## □ list类型支持高效的插入/删除操作

- `l.sort()`: 按list定义的'<'操作符对元素排序.
- `l.sort(cmp)`: 按谓词cmp对元素排序.

## □ <cctype>: 操作字符数据

`isspace(c)`, `isalpha(c)`, `isdigit(c)`, `isalnum(c)`, `ispunct(c)`,  
`isupper(c)`, `islower(c)`, `toupper(c)`, `tolower(c)`

## □ 从C继承而来的头文件——在文件名前加上'c'

e.g. 如果想产生随机数:

```
#include <cstdlib>
```

```
#include <ctime>
```