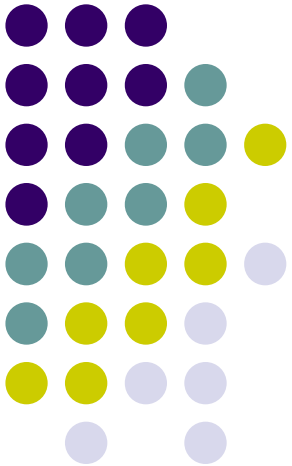
标准库类型

吴清锋

2021年春

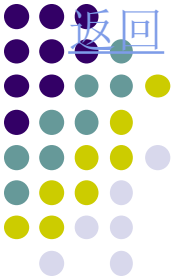


提纲

- 概述
- 标准库string类型
  - 概述
  - string对象的定义和初始化
  - string对象的读写
  - string对象的操作
- 标准库vector类型
  - 概述
  - 类模板
  - vector对象的定义和初始化
  - vector对象的操作
- 迭代器

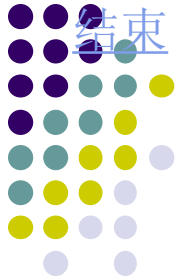
标准库类型，现在的学习目标就是要会掌握并能使用（特别是大量的方法）。在学习完类之后，再来探讨他们背后的原理，究竟是如何设计的！

概述



返回

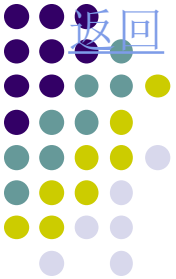
- C++中的数据类型：
  - 基本数据类型(内置数据类型)
  - 构造数据类型
  - 抽象数据类型(程序员自定义数据类型)
- C++中还有**标准库类型**
  - 并非内置在语言中的**数据类型**
  - **标准库类型都定义成类;**
- string和vector是两种最基本的标准库类型
  - string定义了**大小(长度)可变**的**字符串**
  - vector定义了**大小(长度)可变**的**集合**;
- **值得一提：**
  掌握C++的第一步是学习语言的基本知识和标准库



提纲

- 概述——C++的数据类型
- **标准库string类型**
  - 概述
  - string对象的定义和初始化
  - string对象的读写
  - string对象的操作
- 标准库vector类型
  - 概述
  - 类模板
  - vector对象的定义和初始化
  - vector对象的操作
- 迭代器

概述：处理字符串的不同思路



返回

- 借助字符数组来存储和处理字符串
- 借助字符型指针变量来处理字符串
- 使用string类来存储和处理字符串

比较：处理字符串的不同思路

返回

操作	字符数组	字符型指针变量
初始化	**char mystr[]="Hello!";** **char mystr[ ]={"Hello!"};**	**char *p_str="Hello!";**
	"Hello！"字符串常量除了有自己的常量空间外，在数组中也对应存储'H'、'e'、'l'、'l'、'o'、'!'信息。	"Hello！"字符串常量有自己的常量空间，在指针变量p_str中仅仅存储字符串常量的地址信息。
赋 值	**char mystr[81]; mystr="Hello!";** ✗ **strcpy(mystr="Hello!");**	**char *p_str;** **p_str="Hello!";**
	数组名mystr是一指针常量。	指针变量p_str可以指向一字符串常量。
输 入	char mystr[81]; scanf("%s",mystr);	char *p; scanf("%s",p); ✗
	输入的字符串有实际的存储空间。	指针变量p没有字符串的具体存储空间。
字符的修改	**char mystr[]="Hello!"; mystr[0]='h';**	**char *p_str="Hello!";** ***p_str='h';**✗
	对存储在字符数组中的元素进行访问和修改。	无法对常量进行修改。

返回

C++中处理字符数组存储字符串

- 包含C-串处理函数的头文件 #include <cstring>
  如：strcpy、strcat等
- 输入时，cin和getline的区别
  - cin只能读取一个不包含空白字符的字符串

```
char name[100]; //定义C-串的存储空间
cin>>name    //△△△book △△△computer 【Enter】
cout<<name;        book
```

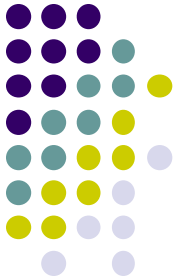
  - cin提供成员函数getline，一次能够读取一行
    即：cin.getline(name,10);
    第一个参数是数组名，第二个参数是待读取的字符个数（包含空字符，留存\0）

返回

标准库string类型概述

- string类型支持**长度可变(可伸缩)**的字符串。
  不是使用字符数组来存储字符串。
- **基本功能**
  使用string类将字符串定义为对象，然后利用string类提供的赋值、连接等字符串操作功能，使得程序员类似处理普通变量一样，方便地实现对字符串的各种处理。
- 与其他标准库类型一样，用户程序要使用string类型对象，必须包含相关头文件：

```
#include <string>
using std::string;
```



```
#include <iostream>
#include <string>
using namespace std;
int main( ) {
    char  charr1[20];  char charr2[20]="Jaguar";
    string str1;  string str2="Panther";
    cout<<"Enter:";              cin>>charr1;
    cout<<"Enter again:";   cin>>str1;
   cout<<charr1<<charr2<<str1<<str2<<endl;
    cout<<"The 3rd letter in"<<charr2<<"is"<<charr2[2]<<endl;
   cout<<"The thrid letter in"<<str2[2]<<"is"<<st[2]<<endl;
   return 0;
}
```

string对象的定义和初始化

- string类对象的定义
  - 定义对象：string **对象名**
    在形式上,与定义变量类似. 例子: string movieTitle;
- 几种初始化string类对象的方式
  - 采用赋值运算符 string movieTitle=“2012”;
  - 采用**“函数调用表示法”**

string s1;	默认构造函数,s1为空串
string s2(s1);	复制构造函数，将s2初始化为s1的一个副本(s1可以是字符数组或string)
string s3(“value”);	将s3初始化为一个字符串字面值副本
string s4(n,’c’);	将s4初始化为字符’c’的n个副本

返回

对象初始化背后有强大的构造函数

- 在创建一个对象时,常常需要作某些初始化的工作。C++提供了构造函数来**处理对象的初始化**;
- 对象的创建过程**自动伴随**着构造函数的调用;
- 构造函数的名字**与类名同名**，而不能由用户任意命名;
- 构造函数是可以**重载**的(同名,参数个数或类型不同)，系统根据函数调用形式去确定对应的构造函数;

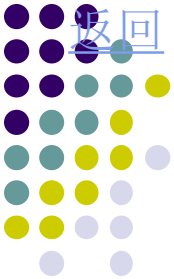
【目前学习，会用善用即可，暂时不用理会具体如何设计构造函数】

string类对象的读写（1）

- 使用标准输入输出函数符来读写string对象
  - 例子

```
string name;
cout<<“What is your name?”;
cin>>name;
cout<<name<<“, 你好” <<endl;
```

  - 在这个程序中，用户输入了一行字符串，这与之前C语言编程不一样，它没有用数组实现具体存储空间的创建，而且字符串的长度没有限制；
  - string类型的输入操作符，特性：
    - 读取并忽略开头所有的空白字符（空格、换行、制表符）；
    - 读取字符直至再次遇到空白字符，读取终止；
      值得一提,这种特性意味着什么?(回忆下scanf())



string类对象的读写（2）

- 用getline读取整行文本

```
getline(cin, str);
```

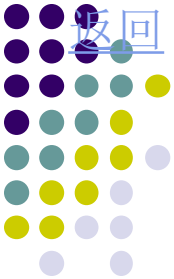
  - 其中：str是string对象
  - 功能：从输入流的下一行读取，并保存读取的内容到string中，但不包括换行符（将换行符转为'\0'）；
  - 若行开头出现换行符，getline并不忽略而是停止读入；

返回

string对象的操作

- string类对象的操作,即实现对字符串进行赋值、连接、复制、查找和交换等功能。
  - 基本形式：**对象名.成员函数　s为string对象**

s.empty()	如果s为空串，则返回true，否则返回false	
s.size()	返回s中字符个数	strlen()
s[n]	返回s中位置为n的字符，位置从0开始计数	
s1+s2	s1,s2连接成一个新字符串	strcat()
s1=s2	把s1内容替换s2的副本	strcpy()
v1==v2	比较v1和V2内容	strcmp()
!=,<,<=,>,>=	保持这些操作符惯有含义	



string对象的连接操作+

- 字符数组存储的字符串，无法直接使用+
- string中的+=：将字符串附加到string对象的末尾
- +或+=（连接）的规则
  - 使用+时，左右操作数必须至少有一个是string类型；

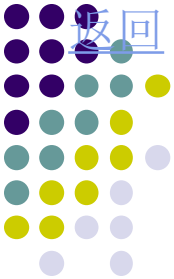
    这是实现+运算符重载的需要！

    string s4=“hello”+”,” 错误

    string s5=s1+”,”+”world”; 正确

    string s6=“hello”+”,”+s2; 错误

返回



string对象的赋值操作

- 字符串常量可以直接赋值给string对象
- string对象可以直接赋值给另一个string对象
- 例子：

```
char charr1[20];    char charr2[20]="Jaguar";
charr1=charr2;  //数组无法直接赋给另一个数组
string str1;            strint str2="Panter";
str1=str2;              //可以
```

配套类型

对比：C-串中的strlen是函数

- size()的返回值类型size_type
  - size()成员函数返回的是string::size_type类型；
  - 许多库类型都定义了一些配套类型，通过配套类型，库类型的使用就能与机器无关。size_type是一种**配套类型**。
  - 值得一提的是，不要把size()返回值赋给一个int变量；

与unsigned具有相同的含义

```
int main() {
   string str1;   str1="volanol";
   for (string::size_type i=0; i != str1.size(); i++)
     {      cout<<str1[i]<<endl;    //s.at(index);
      }
   return 0;
}//为了使用由string类型定义的size_type类型，需要用域操作符来说明size_type类型是由string类定义的
```

返回

string中字符的处理

- string对象中字符的处理——化整为零
  - 从string对象获取字符。例，s[n],n标明要访问字符的位置；前提是string不为空。

```
string str1;
cout<<str1[0]<<endl; //实现不合法，空串长度为0，str1[0]无效
```

  - 常与循环搭配使用（循环变量的类型；循环结束条件）

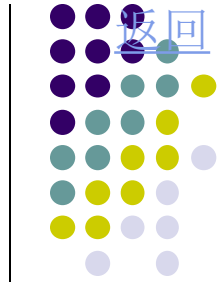
- 字符操作函数适用于对string对象中单个字符(或其他任何的char值)的处理；
  - 这些函数定义在cctype头文件中；

返回

关系运算

- 两个string对象比较时采用大小写敏感的字典序策略
- 练习：输入两个字符串，判断是否为空；若不为空输出长度；比较两个字符串

string的成员函数

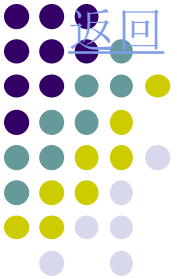


返回

- string中有许多成员函数，为程序员的使用带来方便
- 基本用法：

  对象名.成员函数(参数信息）

- 常用的一些函数
  http://wenku.baidu.com/view/ba2ce41fa300a6c30c229f4d.html



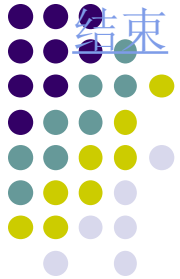
string的优势

- string具有自动调整大小的功能【不需要程序员参与】

  字符数组，总是存在目标数组过小的风险

- string使用很方便，可以直接使用=、+等

  字符数组，需要使用库函数来实现、



提纲

- 概述——C++的数据类型
- 标准库string类型
  - 概述
  - string对象的定义和初始化
  - string对象的读写
  - string对象的操作
- **标准库vector类型**
  - 概述
  - 类模板
  - vector对象的定义和初始化
  - vector对象的操作
- 迭代器

返回

回顾：数组与链表的局限

- （静态内存）数组使用有什么注意的地方？
  - 预先预测并提前限定长度；
  - 数组的比较，需要遍历元素；
- 动态数组
  - 运行阶段设置动态数组的长度
  - 程序员使用“手动”new创建和delete撤销
- 链表使用有什么注意的地方？
  - 操作上显得较为繁琐

返回

vector是动态数组的替代品

- vector模板类是一种更加**健壮**，且有许多附加功能的数组；
- vector类也是使用new和delete来管理内存，但是这种工作是**自动完成**的；
- vector有强大功能，如：
  - 在运行时设置vector的长度
  - 可在末尾附加新数据
  - 提供下标越界检查
  - 提供数组用相等运算和大小比较
  - 提供数组间赋值等运算

标准库vector类型：概述

- vector，从翻译对应的词语： 容器
- vector可以用来存放不同类型的对象（变量）。容器中每个对象都有一个对应的整数索引值。
  - 在数组生存期内，数组的大小是不会改变的，**vector容器则可在运行中动态地增长或缩小；**
- 与其他标准库类型一样，用户程序要使用vector类型对象，必须包含相关头文件：

```
#include <vector>
```

vector类型的声明和定义

- 比喻：要使用 vector 必须指明容器里面存放的是什么类型的对象，就像在坛坛罐罐上面贴上标签告诉别人里面存放的是什么；
- vector不是一种数据类型，而只是一个类模板，必须说明vector中保存的对象的类型，通过将类型放在类模板名称后面的< >来指定类型。
  例子：
  vector<int> ivec;　//内置类型信息
  vector<Sales_item> Sales_vec; //类类型
- 类模板可用来定义任意多种数据类型。
  vector<int>和vector <double>是不种数据类型。

vector对象的定义和初始化（1）

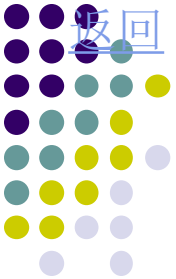
- vector类提供4种构造函数，用来定义且初始化vector对象。
  - 若用T表示数据类型，对象名为v1和v2，则：

vector<T> v1;	定义容器对象v1, 类型为T.默认构造函数v1为空.
vector<T> v1(length);	定义length个元素，元素初始值由T来决定.若int,初始为0.
vector<T> v1(length,a);	定义length个元素,元素初始化为a,a的类型为T.
vector<T> v2(v1);	使用已定义的容器构造新容器.要求v2和v1中必须保存同一种元素类型.

返回

例子：初始化向量对象

- vector<char> A;
  // 容器A为空
- vector<int> B(20);
  //具有20个int的向量，元素初始化值为0
- vector<int> C(20,1);
  vector<string> svec(10,”Hi!”);
  //具有20个int的向量，元素均被置为1
  //可以规定元素个数和元素值实现对vector对象的初始化
- vector<int> D(C);
  //用C初始化D，即D与C一样



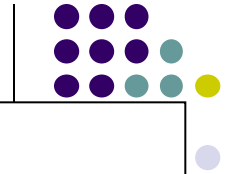
vector对象的长度

- 运行时设置vector的长度
  - 例子：

```
int n;        cin>>n;                    //用户输入长度
vector<char>  vd(n);   //定义n个长度的容器
```

- **vector对象动态增长**
  - vector对象（以及其他标准库容器对象）的重要属性在于可以在运行时高效地添加元素；
  - **更有效的方法：先初始化一个空vector对象，然后再动态地增加元素；**
- 例子
  - 向vector添加元素(push_back())

比较两个容器的例子



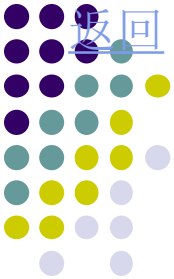
```
#include <vector>
#include <iostream>
int main()
{
        std::vector<int> v1;
        std::vector<int> v2;
        v1.push_back (5);
        v1.push_back (1);
        v2.push_back (1);
        v2.push_back (2);
        v2.push_back (3);
        std::cout << (v1 < v2);
            return 0;
}
```

输出：
0

- 为何是push_back( )?
  而不是v2[0]?
- 容器的直接比较，如果是数组或链表或动态数组，是如何实现比较的？

- 若两容器长度相同、所有元素相等，则两个容器就相等，否则为不等。
- 若两容器长度不同，但较短容器中所有元素都等于较长容器中对应的元素，则较短容器小于另一个容器
- 若两个容器均不是对方的子序列，则取决于所比较的第一个不等的元素

返回



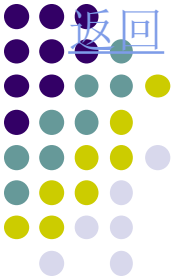
vector对象的操作：概述

- vector类对象的操作
  - 基本形式：**对象名.成员函数**

s.empty()	如果s为空，则返回true，否则返回false	
s.size()	返回s中元素个数，返回值类型为vector类定义的size_type的值	
s[n]	返回s中位置为n的元素	
s.push_back(t)	在s的末尾增加一个值为t的元素	
s1=s2	把s1内容替换s2的副本	这在之前数组的操作(整体操作)中是无法实现的！
v1==v2	比较v1和v2内容	
!=,<,<=,>,>=	保持这些操作符惯有含义	

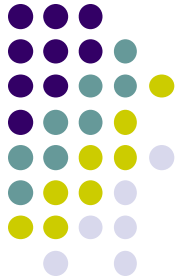
返回

vector的下标运算符和赋值运算符

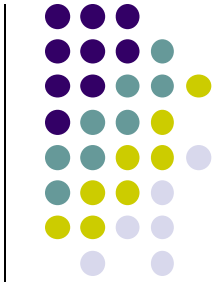


- 下标操作符——化整为零
  - 使用来访问vector容器中的元素;
  - 仅能对确知已存在的元素进行操作;
- 下标操作不添加元素
  - **下标只能用于表达已确定存在的元素**；通过下标操作进行赋值时，不会添加任何元素；
  - 若事先定义vector容器为空或是较小的长度，则必须通过push_back()来实现元素的添加(实现长度的动态扩展);
- 赋值运算符= ——运算符重载的结果
  - vector定义的赋值运算符“=”允许同类型的vector对象相互赋值，而不管它们的长度如何；
  - 它可以改变赋值目标的大小，使它的元素数目与赋值源的元素数目相同;

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
 int main(void) {
   vector<int> ivec;
   for (int i = 0; i < 10; i++) ivec.push_back(i);
   for (vector<int>::size_type x = 0; x != 
   ivec.size(); x++)
        cout << ivec[x] << "\t";
   cout << endl;
   return 0;
}
```



循环终止条件的写法



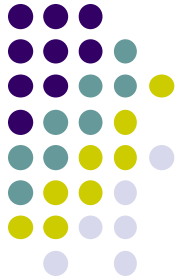
```
for (vector<int>::size_type x = 0; x !=
   ivec.size(); x++)
        cout << ivec[x] << "\t";
```

1 用!=而不是用<来测试vector下标值是否越界

2 在for 语句调用size()成员函数，而不采用事先保存size()的返回值。
有些情况下，vector可能动态增长。

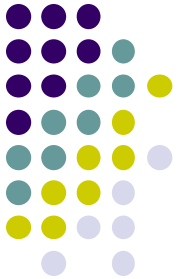
3 联想“内联函数”，size()描述成内联函数，减少在循环过程中调用它的运行代价。

神奇的sting和vector

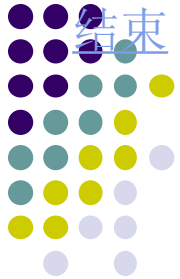


```
#include <iostream>
#include <string>
#include <vector>
using namespace std;
 int main(void) {
    string word;
    vector<string> text;  //定义一个容器，每个对象都是string
    while (cin >> word)
          text.push_back(word);
    return 0;
}
```

测试

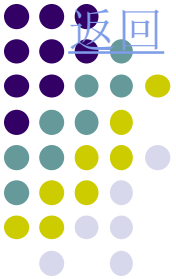


- 通过结构体描述一个学生及选课的基本信息
- 建立学生的报名库



提纲

- 概述——C++的数据类型
- 标准库string类型
  - 概述
  - string对象的定义和初始化
  - string对象的读写
  - string对象的操作
- 标准库vector类型
  - 概述
  - 类模板
  - vector对象的定义和初始化
  - vector对象的操作
- 迭代器



迭代器概述

- vector对象的元素的访问，有两种方式：
  - 使用下标来访问
  - 标准库还提供了另一种访问元素的方法：迭代器；
- 迭代器(遍历器)是一种检查容器内元素并遍历元素的数据类型；
- 标准库为每一种标准容器（包括vector）定义了相应的迭代器类型；迭代器类型提供了比下标操作更通用化的方法。现代C++程序更倾向于使用迭代器而不是下标操作访问容器元素。

容器的iterator类型

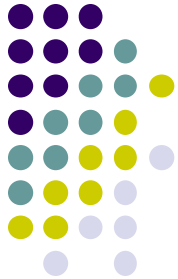
- 每种容器类型都定义了自己的迭代器类型;
- 在vecto容器中，迭代器定义方式：

  vector<type>::iterator 变量名;
- 例子：

  vector<int>::iterator iter;

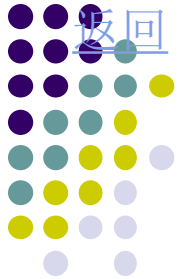
  含义：定义了一个名为iter的变量，它的数据类型是由vector<int>定义的iterator类型；

究竟什么是迭代器



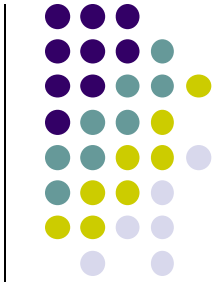
- 迭代器理解为面向对象版本的**指针**
- 可作**解引用**操作来访问元素

这是应用的基础！



迭代器的操作（1）

- begin和end操作——基本操作之一
  - begin(),返回迭代器(指针)，如果容器不为空，该(标记)指向第一个元素【下标为0的元素】；
  - end(),返回迭代器(指针)，该(标记)指向末端元素的下一个（实际指向一个不存在的元素）；它只是一个哨兵的作用，表示我们已处理完了vector中所有元素。
  - 如果vector为空，那么begin()和end()返回的迭代器相同；如果vector不为空，元素存在的范围是半开区间[begin,end)；
- * ——解引用,访问(表示)迭代器所指向的元素
  - 例子：*iter=0;　　//其中,iter是迭代器(指针)变量



```
int main(void) {
    vector<int> ivec;
    for (int i = 0; i < 10; i++)
        ivec.push_back(i);
    vector<int>::iterator  iter; //定义一个迭代器名为iter 的变量
    for (iter = ivec.begin(); iter != ivec.end(); iter++)
        cout << *iter << “\t”; //++迭代器指向下移
    cout << endl;
    return 0;
}
```

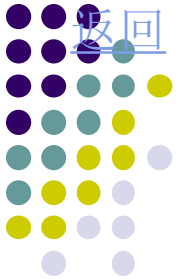
思考：如何间隔一个输出？

对比

下标操作：

```
for(vector<int>::size_type  ix = 0; ix !=
                        ivec.size(); ++ix)
                ivec[ix] = 0 ;
```

前提是有这个空间存在！



迭代器的操作（2）

- 迭代器的算术操作
  注意：可理解成指针变量的算术运算
  - iter+n（iter-n）
    - 运算结果（迭代器），取值范围： [begin(),end()]；
  - iter1-iter2
    - 用于计算两个迭代器对象的距离；
- 迭代器应用的程序示例

```
for (vector<int>::iterator iter=ivec.begin(); iter!=ivec.end();++iter)
        *iter=0;
```

const_iterator（1）

- 使用const_iterator类型定义的迭代器，(迭代器)自身值是可以改变的，但是不能通过该迭代器来改变其所指向的元素的值，**只能用于读取容器内元素；**
- 例子

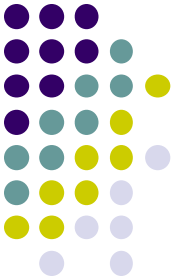
```
for (vector<string>::const_iterator iter=text.begin();
                                    iter!=text.end();++iter)
    *iter="Hello";    //error

for (vector<string>::const_iterator iter=text.begin();
                                    iter!=text.end();++iter)
    cout<<*iter;  //借助iter读取元素值,而不改变值
```

const_iterator（2）

- const_iterator对象与const的iterator对象比较
  - const_iterator对象，它的侧重点在于描述对所指的元素只具有读，而不能具有修改；
  - 定义一个const的迭代器时，必须初始化迭代器；且在运行过程中无法改变(迭代器本身)值；而对于它所指向的元素是否修改没有限制。例子：

```
vector<int> nums(10);
const vector<int>::iterator cit=nums.begin();
*cit=1;        //OK,对指向元素的修改
++cit;         //Error.cit定义时有const修饰符
```

思考和总结：你会什么时候用string和vector



- 当你只有一个字符串的时候，使用string；
- 当你有多个字符串的时候，使用
  vector <string>；
- 当你不确定长度的时候可以用vector，因为定义的时候不需要定义长度，有新元素，通过插入就可增加长度，**还有就是方便**，可以利用库中的几个函数进行操作，如查找，插入操作等，一个函数一个迭代器就可以解决了