一 容器

c++中有两种类型的容器：顺序容器和关联容器，顺序容器主要有：vector、list、deque等。其中vector表示一段连续的内存地址，基于数组的实现，list表示非连续的内存，基于链表实现。deque与vector类似，但是对于首元素提供删除和插入的双向支持。关联容器主要有map和set。map是key-value形式的，set是单值。map和set只能存放唯一的key值，multimap和multiset可以存放多个相同的key值。

容器类自动申请和释放内存，我们无需new和delete操作。

1. 顺序容器

vector

头文件 <vector> **传递参数的时候要传递引用**

|  |  |
| --- | --- |
| 初始化 | vector<int> vec1; //默认初始化，vec1为空  vector<int> vec2(vec1); //使用vec1初始化vec2  vector<int> vec3(vec1.begin(),vec1.end());//使用vec1初始化vec2  vector<int> vec4(10); //10个值为0的元素  vector<int> vec5(10,4); //10个值为4的元素  初始化二维vector：  vector< vector<int> > Matrix(N, vector<int>(M, -1)); |
| 操作方法 | vec1.push\_back(100); //尾部添加元素  int size = vec1.size(); //元素个数  bool isEmpty = vec1.empty(); //判断是否为空  cout<<vec1[0]<<endl; //取得第一个元素  v.insert(v.begin(),8);//在最前面插入新元素。  v.insert(v.begin()+2,1);//在迭代器中第二个元素前插入新元素  v.insert(v.end(),3);//在向量末尾追加新元素。  v.insert(v.end(),4,1);//在尾部插入4个1  vec1.pop\_back(); //删除末尾元素，返回值为void  vec1.erase(vec1.begin(),vec1.begin()+2);//删除vec1[0]-vec1[2]之间的元素，不包括vec1[2]其他元素前移  cout<<(vec1==vec2)?true:false; //判断是否相等==、！=、>=、<=...  vector<int>::iterator iter = vec1.begin(); //获取迭代器首地址  vector<int>::const\_iterator c\_iter = vec1.begin(); //获取const类型迭代器  （这个迭代器是可以自己增加的，但是其所指向的元素是不可以被改变的）  vec1.clear(); //清空元素  vec1.resize(20); // resize() 函数调整出来的 vector 会包含它原先有的那些元素，但是如果你  把空间调小到不足以容纳原来那些元素的程序，那么末尾的元素会被舍弃。  如果 resize() 的空间大于原先的长度，那么新创建的元素会被填充 0 。  // 注意如果你是在调用 resize() 函数后调用 push\_back()，系统还是会分配新的空间来存放添加进来的元素 |
| 遍历 | //下标法  int length = vec1.size();  for(int i=0;i<length;i++)  {  cout<<vec1[i];  }  cout<<endl<<endl;  //迭代器法  vector<int>::iterator iter = vec1.begin();  for(;iter != vec1.end();iter++)  {  cout<<\*iter;  } |

deque <deque>

deque容器类与vector类似，支持随机访问和快速插入和删除，与vector不同，deque还支持从开始端插入数据：push\_front。其余的类似vector操作方法的使用.

list

list是stl实现的双向链表，与向量vector相比，它允许快速的插入和删除，但是随机访问却是比较慢，需要添加头文件#include<list>

|  |  |
| --- | --- |
| 初始化 | list<int> lst1; //创建空list  list<int> lst2(3); //创建含有三个0元素的list  list<int> lst3(3,2); //创建含有三个元素为2的list  list<int> lst4(lst2); //使用lst2初始化lst4  list<int> lst5(lst2.begin(),lst2.end()); //同lst4  list<int> lst5(lst2.begin()+1, lst2.end()); //不可以这样写，因为不存在+，只存在++ |
| 操作方法 | lst1.assign(lst2.begin(),lst2.end()); //分配值,3个值为0的元素，不用管一开始lst1和lst2的长度，assign过  lst1会变的和lst2一样长  lst1.push\_back(10); //末尾添加值 （返回值void）  lst1.pop\_back(); //删除末尾值 (返回值void)  lst1.begin(); //返回首值的迭代器  lst1.end(); //返回尾值后面一元素的迭代器  lst1.clear(); //清空值  bool isEmpty1 = lst1.empty(); //判断为空  lst1.erase(lst1.begin(),lst1.end()); //删除元素  lst1.front(); //返回第一个元素的引用（值）  lst1.back(); //返回最后一个元素的引用（值）  lst1.insert(lst1.begin(),3,2); //从指定位置插入个3个值为2的元素  lst1.rbegin(); //返回第一个元素的前向指针  lst1.remove(2); //相同的元素全部删除  lst1.reverse(); //反转  lst1.size(); //含有元素个数  lst1.sort(); //排序  lst1.unique(); //删除相邻重复元素  \*\*\* 没有取下标操作符 \*\*\* |
| 遍历 | for(list<int>::const\_iterator iter = lst1.begin();iter != lst1.end();iter++)  {  cout<<\*iter;  } |

1. 关联容器

set

头文件 <set>

使用了红黑树

相同的值不存，存进去自动排序好了

set中元素不能修改，只能先删除再增加

|  |  |
| --- | --- |
| 初始化 | set<int> seta; //默认是小于比较器less<int>的set  同前面vector的引用赋值 |
| 操作方法 | |  |  | | --- | --- | | c.size() | 返回当前的元素数量 |     插入：  s.insert(5); //第一次插入5，可以插入  s.insert(5); //第二次插入5，重复元素，不会插入  删除：  s.erase(6); // 即使删除不存在的数也不会报错  检索：  使用find()方法对集合进行检索，如果找到查找的的键值，则返回该键值的迭代器位置；否则，返回集合最后一个元素后面的一个位置，即end()。    计算机生成了可选文字: it = s.find(s); 7 / 查 找 褳 值 为 E 的 元  if(it  C 0  else  C 0  " not fInd    Bound & range：  s.lower\_bound() 返回第一个大于或等于给定关键值的元素 (返回的是迭代器)  s.upper\_bound() 返回第一个大于给定关键值的元素  s.equal\_range() 返回一对定位器，分别表示 第一个大于或等于给定关键值的元素 和 第一个大于给定关键值  的元素，这个返回值是一个pair类型，如果这一对定位器中哪个返回失败，就会等于  s.end()    计算机生成了可选文字: "lower bound & upper bound test ：  < < endl,  “ " 第 一 个 大 于 或 等 于 3 的 元  < 〈 *s ． lower bound(3) < < endl ，  “ " 第 一 个 大 于 或 等 于 2 的 元  < 〈 * s ． 10 勵 er bound(2) 〈 < endl;  “ " 第 一 个 大 于 2 的 元 ： " 〈 0  s ． upper bound(2) < < endl,  equal range test ： " 〈 <  endl,  “ " 第 一 个 大 于 或 等 于 2 的 元  s•equal range(2) .first < 〈 endl,  第 一 个 大 于 2 的 元 ： " “  s•equal range(2)•second < 〈 endl,    判断是否为空：  s.empty()  自定义比较函数：   1. struct mycomp 2. { //自定义比较函数，重载“（）”操作符 3. bool operator() (const int &a, const int &b) 4. { 5. if(a != b) 6. return a > b; 7. else 8. return a > b; 9. } 10. }; 11. set<int,mycomp>::iterator it; 12. for(it = s.begin(); it != s.end(); it++) 13. cout << \*it << " ";     **3.4 赋值**   |  |  | | --- | --- | | **操作** | **效果** | | c1 = c2 | 将c2的元素全部给c1 | | c1.swap(c2) | 将c1和c2 的元素互换 | | swap(c1,c2) | 同上，全局函数 | |
| 遍历 | 正向遍历   * 1. set<**int**>::iterator it; //定义前向迭代器   2. //中序遍历集合中的所有元素   3. **for**(it = s.begin(); it != s.end(); it++)   4. {   5. cout << \*it << " ";   6. }   反向遍历   * 1. set<**int**>::reverse\_iterator rit; //定义反向迭代器   2. //反向遍历集合中的所有元素   3. **for**(rit = s.rbegin(); rit != s.rend(); rit++)   4. {   5. cout << \*rit << " ";   6. } |

multiset

multiset允许元素重复

|  |  |
| --- | --- |
| count (elem) | 返回元素值为elem的个数 |

map

每个关键字只出现一次

也是用红黑树实现的

迭代器可以修改实值，不能修改key

根据key值可以在log（N）时间里快速查找记录

#include <map>

|  |  |
| --- | --- |
| 初始化 | Map<int, string> mapStudent; |
| 操作方法 | 输出：  Iter->first iter->second  插入：  mapStudent.insert(pair<int, string>(1, “student\_one”)); //这个插入如果已经有值了就放弃插入  mapStudent.insert(map<int, string>::value\_type (1, “student\_one”)); //这个插入如果已经有值了就放弃插入  mapStudent[1] =  “student\_one”; //这个插入如果已经有值了就覆盖  如何判断插入成功与否（针对于前两种）：  Pair<map<int, string>::iterator, bool> Insert\_Pair;         Insert\_Pair ＝ mapStudent.insert(pair<int, string>(1, “student\_one”));         If(Insert\_Pair.second == true)         {                Cout<<”Insert Successfully”<<endl;         }         Else         {                Cout<<”Insert Failure”<<endl;         }  大小： 用.size()函数  数据查找：  1 count  2 用find函数来定位数据出现位置，它返回的一个迭代器，当数据出现时，它返回数据所在位置的迭代器，如果  map中没有要查找的数据，它返回的迭代器等于end函数返回的迭代器  3  计算机生成了可选文字: 例 如 ： m 叩 中 己 经 插 入 了 1, 2 ， 3 ， 4 的 话 ， 如 Rlower bound(2) 的 话 ， 返 回 的 2 ， 而 upper-bound （ 2 ） 的 话 ， 返 回 的 就 是 3  Equal_range 函 数 返 回 一 个 pair' pair 里 面 第 一 个 变 量 是 Lower bound 返 回 的 迭 代 器 ， pair 里 面 第 二 个 迭 代 器 是 Upper_bound 返  回 的 迭 代 器 ， 如 果 这 两 个 迭 代 器 相 等 的 话 ， 则 说 明 m 叩 中 不 出 现 这 个 关 键 字 ， 程 序 说 明    清空：clear()  判空：empty()  删除：erase() 三个重载  计算机生成了可选文字: 蜘 果 要 册 刂 除 用 迭 代 器 册 刂 除  mapC1nt: strm?::lterator Iter;  iter = mapStudent.find(l);  mapStudent-erase(iter);  蜘 果 要 册 刂 除 ， 用 关 键 字 册 刂 除  Intn—mapstudent-erase(l);/äükff 刂 除 了 会 返 回 ， 否 则 返 回  用 迭 代 器 ， 成 片 的 册 刂 除  一 下 代 码 把 个 ． m 胄 空  0  mapStudent-earse(mapStudent.begm(): mapStudent.end());  成 片 册 刂 除 要 注 意 的 是 ， 也 是 TL 的 特 性 ， 册 刂 除 区 间 是 一 个 前 闭 后 开 的 集 合    自定义比较操作符：  计算机生成了可选文字: Typedef struct tagStudentlnfo  Int nID•  String strName;  BOOI operator < (tagStudentlnfo const& A) const  这 个 函 数 指 定 排 序 策 略 ， 按 非 序 ， 如 果 n1D 相 等 的 话 ， 按 “ N 灬 。 排 序  If(nIDc A-nID) returntrue:  If(nlD— A-nlD)return < 0 ：  士 串  Return false;  )Studentlnfo: *PStudentlnfo  学 生 亻 臼 |
| 遍历 | 正向：  map<int, string>::iterator  iter;         for(iter = mapStudent.begin(); iter != mapStudent.end(); iter++)  {         Cout<<iter->first<<”   ”<<iter->second<<end;  }    倒向：  map<int, string>::reverse\_iterator iter;  for (iter = mapStudent.rbegin(); iter != mapStudent.rend(); iter++)  {  cout << iter->first << " " << iter->second;  cout << endl;  } |

multimap

|  |
| --- |
| 初始化 multimap <string, string> DNS\_daemon; |
| 插入 DNS\_daemon.insert(make\_pair("213.108.96.7","cppzone.com")); //再插依然可以插进去 |
| * 1. //单独遍历只有涉及某个key, equal\_range(key)   2. cout << endl << "-----------------visit only one key multimap--------------- " << endl;   3. pair<Iterator,Iterator> it2;   4. it2 = mump.equal\_range(1);   5. /\*   6. equal\_range()返回的是一个pair，该pair的存放的是两个迭代器   7. equal\_range().first 存放的迭代器指向的是起始位置   8. equal\_range().second 存放的迭代器指向的是末尾的位置   9. 有点类似于map中的begin()/end()   10. \*/   12. **for**(it = it2.first;it != it2.second;++it) {   13. cout << it->first << " " << it->second << endl;   15. }   16. cout << " ------------------- End ----------------- " << endl;        * 1. //删除某个键值，某个值   3. cout << " -----------Delete only one key and only one value ------------- " << endl;   4. it2 = mump.equal\_range(1);   5. **for**(it = it2.first;it != it2.second; ++it) {   6. **if**(it->first == 1 && it->second == 5) {   7. cout << "delete: " << it->first << " " << it->second << endl;   8. mump.erase(it);   9. }   10. }   12. it = mump.begin();   13. **for**(;it != mump.end();++it) {   14. cout << it->first << " " << it->second << endl;   15. }   16. cout << "-------------------- End --------------------------- " << endl;        * 1. //删除某个键值   2. cout << endl << "-------------Delete only one key------------ " << endl;   3. it = mump.find(1);   4. cout << " delete 3 after " << endl;   5. mump.erase(3);   6. it = mump.begin();   7. **for**(;it != mump.end();++it) {   8. cout << it->first << " " << it->second << endl;   9. }   10. cout << "-------------End----------------" << endl;     来自 <<https://blog.csdn.net/zhongshijunacm/article/details/41651815>> |

pairs

pair将一对值组合成一个值，这一对值可以具有不同的数据类型（T1和T2，也可以是对象），两个值可以分别用pair的两个公有函数first和second访问。

|  |  |
| --- | --- |
| 初始化 | * 1. pair<**int**, **double**> p1;  //使用默认构造函数   2. pair<**int**, **double**> p2(1, 2.4);  //用给定值初始化   3. pair<**int**, **double**> p3(p2);  //拷贝构造函数 |
| 操作方法 | 访问两个元素：  p1.first p1.second  make\_pair:  p1 = make\_pair(1, 1.2);  超过两个元素：  计算机生成了可选文字: LcppJ  paIr < 1 nt ， pairKInt Int 》 》  写 法 极 惴 ． （ 后 边 的 两 个 > > 要 有 空 恪 否 则 就 会 是 > > 位 移 运 算 符 ） |

1. 容器适配器

stack

头文件 <stack>

stack 不提供元素的任何迭代器操作，因此，stack 容器也就不会向外部提供可用的前向或反向迭代器类型

|  |  |
| --- | --- |
| 初始化 | 使用默认的 deque 为底层容器，创建一个空的堆栈对象 s  stack<int>  s;  利用 s1 ，创建一个以双向链表为底层容器的空堆栈对象 s2 （复制构造）  stack<int, list<int> >   s2(s1); |
| 操作方法 | 入栈  void  push(const value\_type& x)  出栈  void pop()  全部出栈       while(!s.empty())       {           s.pop();// 出栈       }  取栈顶  value\_type&  top()  判空  bool empty()    size() 返回栈中元素数目 |

queue

头文件 <queue>

|  |  |
| --- | --- |
| 初始化 | queue<int> q1; |
| 操作方法 | 入栈  void  push(const value\_type& x)  出栈  void pop()  全部出栈       while(!s.empty())       {           s.pop();// 出栈       }  取队首  value\_type&  front()  取队尾  value\_type&  back()  判空  bool empty()    size() 返回栈中元素数目 |

priority\_queue

头文件 <queue>

priority\_queue 模板类有三个模板参数，第一个是元素类型，第二个容器类型，第三个是比较算子。其中后两个都可以省略，默认容器为vector，默认算子为less，即小的往前排。

示例定义：

priority\_queue<int> q1;

priority\_queue< pair<int, int> > q2; // 注意在两个尖括号之间一定要留空格。

priority\_queue<int, vector<int>, greater<int> > q3; // 定义小的先出队

示例程序：

计算机生成了可选文字:
class T 
public 
int X, y, 4 
T(int a, int b, int c):x(a), Y(b), z(c) 
bool operator < (const T &tl ， const T &t2) 
return tl .z < t2 ． 4 / / 按 照 z 的 顺 序 来 决 定 tl 和 t2 的 顺 序 
main() 
priority q; 
q push(T(4,4,3)); 
qpush(T(2,2,5)); 
qpush()( 1 ， 5 ， 4)) ； 
qpush(T(3,3,6)); 
while (!q.empty()) 
T t = q ． t 叩 0 ； q POP(); 
return 1' 

1. 其他

string

头文件 <string>

|  |  |
| --- | --- |
| 初始化 | 1 string s;//默认初始化，一个空字符串  2 string s1("ssss");//s1是字面值“ssss”的副本  3 string s2(s1);//s2是s1的副本 （深度拷贝）  4 string s3=s2;//s3是s2的副本 （深度拷贝）  5 string s4(10,'c');//把s4初始化  6 string s5="hiya";//拷贝初始化  7 string s6=string(10,'c');//拷贝初始化，生成一个初始化好的对象，拷贝给s6 //string s(cp,n)  8 char cs[]="12345"; string s7(cs,3);//复制字符串cs的前3个字符到s当中 //string s(s2,pos2) （这个无所谓越界问题）  9 string s8="asac";  string s9(s8,2);//从s2的第二个字符开始拷贝，不能超过s2的size //string s(s2,pos2,len2)  10 string s10="qweqweqweq";  string s11(s10,3,4);//s4是s3从下标3开始4个字符的拷贝，超过s3.size出现未定义 |
| 操作方法 | 1 substr 注意substr没有迭代器作为参数的操作  string s="abcdefg";  //s.substr(pos1,n)返回字符串位置为pos1后面的n个字符组成的串  string s2=s.substr(1,5);//bcdef  //s.substr(pos)//得到一个pos到结尾的串  string s3=s.substr(4);//efg    2 insert  s.insert(pos,str)//在s的pos位置插入str  s.insert(pos,str,a,n)在s的pos位置插入str中插入位置a到后面的n个字符  s.insert(pos,cstr,n)//在pos位置插入cstr字符串从开始到后面的n个字符  s.insert(pos,cstr)在s的pos位置插入cstr  s.insert(pos,n,ch)在s.pos位置上面插入n个ch  it = str.insert(str.begin() + 5, ','); // s.insert(s.it,ch)在s的it指向位置前面插入一个字符ch，返回新插入的位置的迭代器  // to be, not to be 对于逗号，这是begin()+5, \*it == ','    // to be, not to be: that is the question...  s.insert(it,str.ita,str.itb) 在it所指向的位置的前面插入[ita,itb)的字符串  (str.insert(it + 2, str3.begin(), str3.begin() + 3); // to be, or not to be: that is the question…)    3 erase  删除操作有三种   * 指定pos和len，其中pos为为起始位置，pos以及后面len-1个字符串都删除 * 迭代器，删除迭代器指向的字符 * 迭代器范围，删除这一范围的字符串，范围左闭右开   计算机生成了可选文字: int main 0  Std ： ： string Str （ "This is an example sentence. " ）  Std ： ： cout （ < 《 < 'Nn '  7 ／ " T ， 15 15 E n example sentence ．  Str. （ la ， 8 ）  7 百 掊 指 定 删 除 的 享 符 畢 位 置 第 十 个 后 面 的 8 个 享 符  Std ： ： cout （ < 《 < ' In ．  7 " 1 ， IS 15 an sentence ·  str. erase (str.begin()+9);//  除 迭 代 器 指 向 的 享 符  Std ： ： cout （ < 《 < ' In ．  Str. (Str.begin()+S,  打 删 除 迭 代 器 范 圃 的 享 符  Std ： ： cout （ < 《 < ' In ．  7 " 1 ， IS 15 a sentence ．  r 止 n 匕 0 一 ；  4 append & replace  append可用+=重载，replace是先删除后增加  std::string str; std::string str2="Writing "; std::string str3="print 10 and then 5 more";  //直接追加一个str2的字符串  str.append(str2); // "Writing "  //后面追加str3第6个字符开始的3个字符串  str.append(str3,6,3); // "10 "  //追加字符串形参的前5个字符  str.append("dots are cool",5); // "dots "  //直接添加 str.append("here: "); // "here: "  //添加10个'.' str.append(10u,'.'); // ".........."  //添加str3迭代器范围的字符串 str.append(str3.begin()+8,str3.end()); // " and then 5 more"    replace支持使用无符号整数寻找位置，也支持用迭代器寻找位置  计算机生成了可选文字: St 匕 ． ： strlng  St 匕 ． ： strlng  St 匕 ． ： strlng  St 匕 ． ： strlng  "this 15 a test strlns.  str2z"n example"  Ste:"sample phrase"  Str4z"use-FLJI ．  / 7 replace used •n the same order a S 0e5 〔 rl 匕 e 匕 above ：  / 7 LISIng POS 二 t10n5 ：  St 匕 ： ： strlng Str:bESe;  第 I 享 符 以 及 后 面 的 4 个 享 符 r 玳 替  Str. ， r2 〕 ；  e1234557B 目 “ 1234557B 目 “ 12345  "th1S 15 E test stung ·  "th1S 15 E n example strl.ng ．  第 个 享 符 畢 以 及 后 面 的 享 符 用 str 的 第 7 个 享 符 以 及 后 面 的 5 个 享 符 代 替  7 19  5 丿 St 「 3 ， 7 ， 5 ） ；  th1S 15 E n example phrase ．  行 第 8 个 享 符 以 及 后 面 的 g 个 享 符 用 享 符 畢 黲 數 代 替  Str. la 丿 "just a");  th1S 15 Dust E phrase ·  行 第 8 个 享 符 以 及 后 面 的 5 个 享 符 用 享 符 畢 黲 數 的 前 7 个 享 符 替 换  二 8 ， ， 5h0 「 t 丿 巪 7 ）  第 以 及 后 面 的 砰 享 符 用 3 个 叹 号 替 换  7 22  Str. 〔 22 1 ， 3 ，  7 迭 代 器 的 原 理 同 上  7 LISIng Iterators ：  th1S 15 E Short phrase ．  "th1S 15 E Short phrase ．  234557B9 “ 12345E7B9 “  sample phrase ·  Str.  Str.  Str.  Str.  Str.  巪 7  I replace  I replace  I replace  ／ " replace  phrase.  = 00  = 00001 ．  IS useful ．  · ） 8 ， ． "is = 001ne 5  · ） 11 ， Str 丿 ． begin() ，      5 assign  计算机生成了可选文字: St 匕 ． ： strlng Str;  St 匕 ． ： strlng base:"The quick 匕 「 WI fox Jumps over a IE ' 匕 OS ．  7 used 1 n the same order as above ：  7 百 掊 把 匕 ” e 值 给  Str.ESSign(bESe);  行 把 “ 第 1 个 享 符 以 及 后 面 的 E 个 享 符 给 r  把 黲 數 中 的 刂 6 个 享 符 畢 给 r  pangrams are = 001  百 掊 使 用 黲 數 值  str.assign("c-string")  7 给 st 值 ” 个 ” ． 享 符  值 是 个  指 定 “ 迭 代 器 范 圃 的 享 符 畢  7 此  " 匕 n fox"  pangram"  "c-strlng"  一 12 ） ；  " +0 Jumps over  · 15 ， ． end 0  7  6 find  find函数主要是查找一个字符串是否在调用的字符串中出现过  //在str当中查找第一个出现的needle，找到则返回出现的位置，否则返回结尾  std::size\_t found = str.find(str2);  if (found!=std::string::npos)  std::cout << "first 'needle' found at: " << found << '\n';  //在str当中，从第found+1的位置开始查找参数字符串的前6个字符 （如果只有一个数字，是起始位置）  found=str.find("needles are small",found+1,6);  if (found!=std::string::npos)  std::cout << "second 'needle' found at: " << found << '\n';  //查找一个字符  found=str.find('.');  // let's replace the first needle:  str.replace(str.find(str2),str2.length(),"preposition");  7 rfind  rfind函数就是找最后一个出现的匹配字符串，返回的位置仍然是从前往后数的。  8 find\_of   * find\_first\_of(args) **查找args中任何一个字符第一次出现的位置** * find\_last\_of(args) **最后一个出现的位置** * find\_fist\_not\_of(args) **查找第一个不在args中的字符** * find\_last\_not\_of **查找最后一个不在args中出现的字符**     std::size\_t found1 = str1.find\_first\_of("aeiou");  //把所有元音找出来用\*代替  while (found1!=std::string::npos) { str1[found1]='\*'; found1=str1.find\_first\_of("aeiou",found1+1); }    9 compare  如果两个字符串相等，那么返回0，调用对象大于参数返回1，小于返回-1  由于string重载了运算符，可以直接用>,<，==来进行比较    计算机生成了可选文字: if （ ．  15 not “ 《 （ Str2 < <  行 strl 的 第 E 个 享 符 以 及 后 面 的 ' 个 享 符 和 黲 數 比 较  乁 酽  if （ 意 rI ． （ 5 ， 5 ， ' ' a001e " ）  St 匕 ． ： = 00t 《 < " 5t111 ，  15 an B001e 、 n "  if （ 意 r2 ． ） 一 5 ， 5 ， ''apple" ）  St 匕 ． ： = 00t 《 < "and " 《 < Str2 < <  15 al 0 an  行 strl 的 第 E 个 享 符 以 及 后 面 的 ' 个 享 符 和 str2 的 第 ' 个 享 符 以 及 后 面 的 4 个 享 符 比 较  if （ 意 rI ． 匚 （ 5 ， 5 ， ， ， 5 ）  St 匕 ： ： = 00t 《 < ''therefore, both    10 字符指针和string的转换    计算机生成了可选文字: void 0  strlng 51  "aaabbb"  cout 《 < Sl · c_str( ） < < en 匕 1 ；  幬 贝 3 个 享 符 ， 从 1 个 享 符 开 始 位 置 下 标 从 开 始 ）  Char bufC 12B ]  51 ． L00 ， ' （ 匕 uf ， 3  = 00t < < ''buf ：  = run;  < < buf  11 数值转换   |  |  | | --- | --- | | **string和数值转换** |  | | to\_string(val) | 把val转换成string | | stoi(s,p,b) | 把字符串s从p开始转换成b进制的int (p可以就是一个整数) | | stol(s,p,b) | long | | stoul(s,p,b) | unsigned long | | stoll(s,p,b) | long long | | stoull(s,p,b) | unsigned long long | | stof(s,p) | float | | stod(s,p) | double | | stold(s,p) | long double | |
| 遍历 | 计算机生成了可选文字: 1 數 组 方 贰  for(int i  e; （ 51 ． leng 意 h （ ） ；  二 t ÷  《 < endI,  2 迭 代 器  for( string ： ： iterator it  It < <  《 < endI,  Sl.begin();  = ． en 匕 0 |

二 算法

1 copy

2 copy\_backward

3 fill

4 fill\_n

5 generate

6 iter\_swap

7 move

8 remove

9 unique

10 find

11 find\_if

12 count

13 count\_if

14 search

15 search\_n

16 find\_end

17 transform

18 sort

三 迭代器 iterator

正向遍历 iter = i.begin() ; != i.end()

逆向遍历 iter = i.rbegin(); != i.rend()

计算机生成了可选文字:
只 有 顺 序 容 器 和 关 联 容 器 支 持 法 代 器 遍 历 ， 各 容 器 支 持 的 法 代 器 的 类 别 如 下 
容 器 
deque 
set 
m ultiset 
map 
multimap 
stack 
ql_lel_le 
prlOrlty_qLleLle 
支 待 的 
迭 代 器 类 别 
随 机 访 
随 机 访 
双 向 
双 向 
双 向 
双 向 
双 向 
不 支 恃 
不 支 恃 
不 支 恃 
说 明 
一 种 随 机 访 问 的 数 类 型 ， 提 供 了 对 数 组 元 寮 进 行 快 速 随 机 访 问 以 及 在 序 列 尾 部 进 
行 快 速 的 届 入 和 亂 除 臊 作 的 功 能 。 可 以 冉 雩 要 的 时 候 修 改 其 自 身 的 大 小 
一 种 随 机 访 问 的 数 类 型 ， 提 供 了 序 列 两 端 快 速 进 行 届 入 和 删 除 操 作 的 功 能 。 可 以 
冉 雪 要 的 时 候 修 改 其 自 身 的 大 小 
一 种 不 支 待 随 机 访 问 的 数 組 类 型 ， 届 入 和 删 除 所 花 的 时 间 是 固 定 的 ， 与 位 置 无 
关 。 
一 种 随 机 荐 取 的 容 器 ， 其 关 字 和 数 啹 元 素 是 同 一 个 值 。 所 有 元 素 都 必 须 具 有 惟 一 
值 。 
一 种 随 机 荐 取 的 容 器 ， 其 关 字 和 数 啹 元 素 是 同 一 个 值 。 可 以 包 飠 重 复 的 元 素 。 
一 种 包 含 成 对 数 值 的 容 器 ， 一 ， 廬 是 实 际 数 啹 值 ， 另 一 个 是 用 来 寻 找 数 啹 的 关 键 
子 。 一 个 特 定 的 关 键 字 只 能 与 一 个 元 寮 关 联 。 
一 种 包 含 成 对 数 值 的 容 器 ， 一 ， 廬 是 实 际 数 啹 值 ， 另 一 个 是 用 来 寻 找 数 啹 的 关 键 
。 一 个 关 隧 字 可 以 与 多 个 数 啹 元 素 关 联 。 
适 配 器 容 器 类 型 
适 配 器 容 器 类 型 
适 配 器 容 器 类 型 
#vector ， de ue 或 list 对 象 创 建 了 一 个 先 进 后 出 容 器 
用 deque list 对 象 创 建 了 一 个 先 进 先 出 容 器 
用 vector ． de ue 对 象 创 建 了 一 个 排 序 队 列 

Reference:

0. 核心官网 <http://www.cplusplus.com/reference>

1. ACM: <http://www.yichenxing.com/>
2. STL: <https://blog.csdn.net/runninglion/article/details/77744681>
3. <https://www.cnblogs.com/cxq0017/p/6555533.html>
4. <https://blog.csdn.net/hero_myself/article/details/52304794>
5. const迭代器 和 const\_iterator的区别 <https://blog.csdn.net/wmn_wmn/article/details/8144151>
6. string的操作 <https://blog.csdn.net/Duke10/article/details/78344226>

<https://blog.csdn.net/tengfei461807914/article/details/52203202>

7. set： <https://blog.csdn.net/cerberux/article/details/51774121>

<https://blog.csdn.net/ac_hexin/article/details/52042562>

<https://www.cnblogs.com/ChinaHook/p/6985444.html>

8. map： <https://blog.csdn.net/bat603/article/details/1456141>

9. multimap： <https://blog.csdn.net/believefym/article/details/1627874>

<https://blog.csdn.net/zhongshijunacm/article/details/41651815>

10. stack: <https://blog.csdn.net/lyj2014211626/article/details/66967761>

11 queue: <https://www.cnblogs.com/qigaohua/p/5803730.html>

12 算法： <https://blog.csdn.net/robin__chou/article/details/53204970>

<https://blog.csdn.net/vast_sea/article/details/8152443>

<https://blog.csdn.net/hihozoo/article/details/50804202>

13 迭代器： <https://blog.csdn.net/qq_23100787/article/details/51388163>