

## 知识点1【STL的概述】

STL的三大组件：容器(container) 算法(algorithm) 迭代器(iterator)。

STL:六大组件

算法的分类：

迭代器的分类：

## 知识点2【迭代器的案例】

案例：容器vector

深入了解for\_each

案例2：容器也可以存放自定义数据类型

案例3：容器嵌套容器（了解）

## 知识点3【string类】

1、案例：string的构造 和 赋值

2、案例：string的字符的存取（注意）

3、案例:字符串拼接2-1

4、案例：字符串的查找替换

5、字符串比较

6、字符串提取

7、字符串的插入删除2-2

8、string 和c风格的字符串转换

## 知识点4【vector容器】单端动态数组

1、vector容器的概述

2、vector的容量capacity和大小size的区别（了解）

3、vector另寻地址的次数（了解）

4、vector的未雨绸缪机制（了解）

5、vector的构造函数

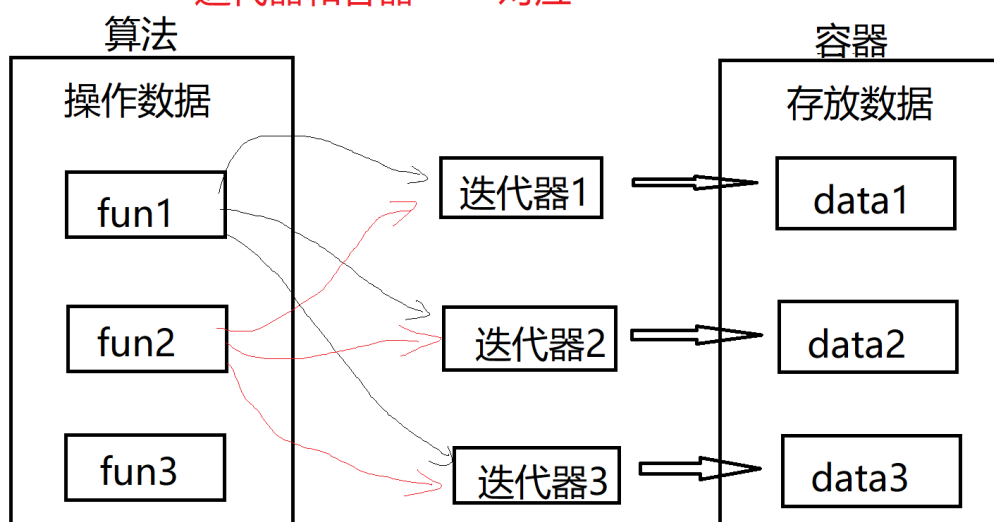
## 知识点1 【STL的概述】

STL(Standard Template Library,标准模板库)

**STL的三大组件：容器(container) 算法(algorithm) 迭代器(iterator)。**

算法操作数据 容器存储数据 迭代器是算法操作容器的桥梁

迭代器和容器 一一对应



## STL:六大组件

容器 算法 迭代器 仿函数 适配器 空间配置器

容器：存放数据

算法：操作数据

迭代器：容器和算法的桥梁

仿函数：为算法 提供更多的策略

适配器：为算法 提供更多的参数接口

空间配置器：管理容器和算法的空间

## 算法的分类：

质变算法：是指运算过程中会更改区间内的元素的内容。例如拷贝，替换，删除等等

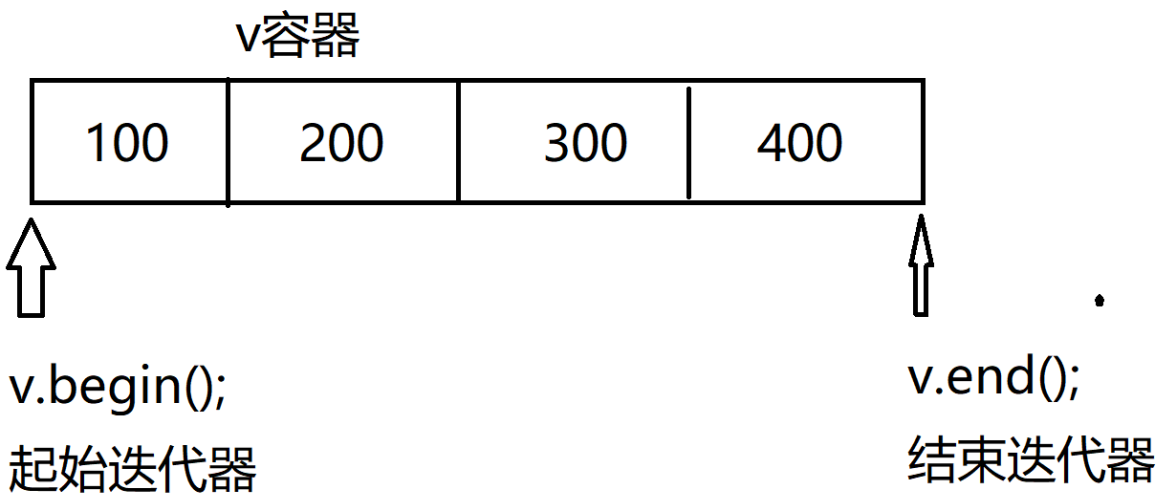
非质变算法：是指运算过程中不会更改区间内的元素内容，例如查找、计数、遍历、寻找极值等等

# 迭代器的分类：

迭代器的种类:

输入迭代器	提供对数据的只读访问	只读，支持++、==、!=
输出迭代器	提供对数据的只写访问	只写，支持++
前向迭代器	提供读写操作，并能向前推进迭代器	读写，支持++、==、!=
双向迭代器	提供读写操作，并能向前和向后操作	读写，支持++、--，
随机访问迭代器	提供读写操作，并能以跳跃的方式访问容器的任意数据，是功能最强的迭代器	读写，支持++、--、[n]、-n、<、<=、>、>=

## 知识点2 【迭代器的案例】



起始迭代器： 指向第0个元素的位置

结束迭代器： 尾元素的下一个元素的位置

### 案例： 容器vector

```
1 #include <iostream>
2 #include<vector>
3 #include<algorithm>
4 using namespace std;
5 void myPrintInt(int val);
6 void test01()
7 {
```

```

8 //单端动态数组vector 类模板
9 vector<int> v;//v就是一个具体的vector容器
10
11 //push_back 尾部插入
12 v.push_back(100);
13 v.push_back(200);
14 v.push_back(300);
15 v.push_back(400);
16
17 //访问数据
18 //定义一个迭代器存储 v的起始迭代器
19 vector<int>::iterator beginIt = v.begin();
20 //定义一个迭代器存储 v的结束迭代器
21 vector<int>::iterator endIt = v.end();
22
23 //for循环遍历1
24 for(;beginIt != endIt; beginIt++)
25 {
26 //对迭代器取* 代表的是 容器的元素
27 //*beginIt
28 cout<<*beginIt<<" ";
29 }
30 cout<<endl;
31
32 //for循环遍历2（推荐）
33 for(vector<int>::iterator it=v.begin(); it !=v.end(); it++)
34 {
35 cout<<*it<<" ";
36 }
37 cout<<endl;
38
39 //STL提供的算法来遍历容器(包含算法头文件 algorithm)
40 //for_each 从容器的起始--->结束 逐个元素取出
41 //myPrintInt 容器数据的打印方式
42 for_each(v.begin(), v.end(), myPrintInt);
43 cout<<endl;
44 }
45
46 void myPrintInt(int val)
47 {

```

```

48  cout<<val<<" ";
49  }
50
51  int main(int argc, char *argv[])
52  {
53      test01();
54      return 0;
55  }
56

```

运行结果：

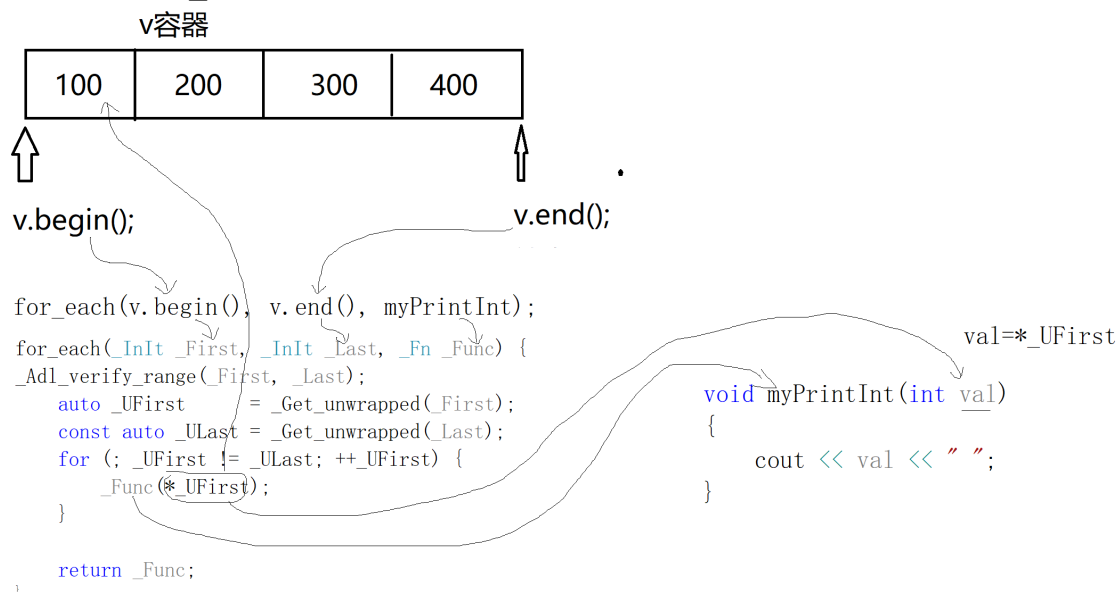
C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator\_process\_stub.exe

```

100 200 300 400
100 200 300 400
100 200 300 400

```

## 深入了解for\_each



## 案例2：容器也可以存放自定义数据类型

```

1 void myPrintPerson(Person &ob)
2 {
3     cout<<"name = "<<ob.name<<" , age = "<<ob.age<<endl;
4 }
5 void test02()
6 {
7     Person ob1("德玛", 18);
8     Person ob2("小炮", 28);
9     Person ob3("小法", 38);
10    Person ob4("小智", 48);
11
12    //定义一个vector容器存放ob1~ob4的数据
13    vector<Person> v;
14    v.push_back(ob1);
15    v.push_back(ob2);
16    v.push_back(ob3);
17    v.push_back(ob4);
18
19    for_each(v.begin(), v.end(), myPrintPerson);
20 }
21
22 int main(int argc, char *argv[])
23 {
24     test02();
25     return 0;
26 }

```

运行结果：

```

name = 德玛, age =18
name = 小炮, age =28
name = 小法, age =38
name = 小智, age =48

```

### 案例3：容器嵌套容器（了解）

```

1 void test03()

```

```

2 {
3     vector<int> v1;
4     vector<int> v2;
5     vector<int> v3;
6
7     v1.push_back(10);
8     v1.push_back(20);
9     v1.push_back(30);
10    v1.push_back(40);
11
12    v2.push_back(100);
13    v2.push_back(200);
14    v2.push_back(300);
15    v2.push_back(400);
16
17    v3.push_back(1000);
18    v3.push_back(2000);
19    v3.push_back(3000);
20    v3.push_back(4000);
21
22    //需求在定义一个vector容器 存放 v1 v2 v3
23    vector<vector<int>> v;
24    v.push_back(v1);
25    v.push_back(v2);
26    v.push_back(v3);
27
28    //for循环遍历
29    for(vector<vector<int>>::iterator it = v.begin(); it!=v.end(); it++)
30    {
31        /*it == vector<int> v1 v2 v3
32        for(vector<int>::iterator mit=(*it).begin(); mit!=(*it).end(); mit++ )
33        {
34            /*mit ==int
35            cout<<*mit<<" ";
36        }
37        cout<<endl;
38    }
39 }

```

运行结果：

10 20 30 40  
100 200 300 400  
1000 2000 3000 4000

## 知识点3 【string类】

### 1、案例：string的构造 和 赋值

```
1  /*
2  3.1.2.1 string 构造函数
3  string();//创建一个空的字符串 例如: string str;
4  string(const string& str);//使用一个string对象初始化另一个string对象
5  string(const char* s);//使用字符串s初始化
6  string(int n, char c);//使用n个字符c初始化 v
7
8  3.1.2.2 string基本赋值操作
9  string& operator=(const char* s);//char*类型字符串 赋值给当前的字符串
10 string& operator=(const string &s);//把字符串s赋给当前的字符串
11 string& operator=(char c);//字符赋值给当前的字符串
12 string& assign(const char *s);//把字符串s赋给当前的字符串
13 string& assign(const char *s, int n);//把字符串s的前n个字符赋给当前的字符串
14 string& assign(const string &s);//把字符串s赋给当前字符串
15 string& assign(int n, char c);//用n个字符c赋给当前字符串
16 string& assign(const string &s, int start, int n);//将s从start开始n个
17 */
18 void test01()
19 {
20     //string(const char* s);//使用字符串s初始化
21     string str1("hello string");
22     cout<<str1<<endl;//"hello string"
23
24     //string(int n, char c);//使用n个字符c初始化
25     string str2(10, 'H');
26     cout<<str2<<endl;//"HHHHHHHHHHH"
```



```
27
28 string str3 = str2;
29 cout<<str3<<endl;//"HHHHHHHHHH"
30
31 string str4;
32 //string& operator=(const string &s);//把字符串s赋给当前的字符串
33 str4 = str1;
34 cout<<str4<<endl;//"hello string"
35
36 //string& operator=(const char* s);//char*类型字符串 赋值给当前的字符串
37 string str5;
38 str5 = "hello str5";
39 cout<<str5<<endl;//"hello str5"
40
41 //string& operator=(char c);//字符赋值给当前的字符串
42 string str6;
43 str6 = 'H';
44 cout<<str6<<endl;//"H"
45
46 //string& assign(const char *s);//把字符串s赋给当前的字符串
47 string str7;
48 str7.assign("hello str7");
49 cout<<str7<<endl;//"hello str7"
50
51 //string& assign(const char *s, int n);//把字符串s的前n个字符赋给当前的字符串
52 string str8;
53 str8.assign("hello str8", 5);
54 cout<<str8<<endl;//"hello"
55
56 //string& assign(const string &s);//把字符串s赋给当前字符串
57 string str9;
58 str9.assign(str8);
59 cout<<str9<<endl;//"hello"
60
61 //string& assign(int n, char c);//用n个字符c赋给当前字符串
62 string str10;
63 str10.assign(10, 'W');
64 cout<<str10<<endl;//"WWWWWWWWWW"
65
66 //string& assign(const string &s, int start, int n);//将s从start开始n个
```

```
67 string str11;
68 str11.assign("hehehahahaxixi", 4, 6);
69 cout<<str11<<endl;//"hahaha"
70 }
```

## 2、案例：string的字符的存取（注意）

```
1  /*
2  3.1.2.3 string存取字符操作
3  char& operator[](int n);//通过[]方式取字符
4  char& at(int n);//通过at方法获取字符
5  */
6
7  void test02()
8  {
9      string str1="hello string";
10     cout<<str1[1]<<endl;//'e'
11     cout<<str1.at(1)<<endl;//'e'
12
13     str1[1]='E';
14     cout<<str1<<endl;//"hEllo string"
15     str1.at(7) = 'T';
16     cout<<str1<<endl;//"hEllo sTring"
17
18     //[]和at的区别
19     try
20     {
21         //str1[1000]='G';//越界 []不抛出异常
22         str1.at(1000)='G';//越界 at会抛出异常
23     }
24     catch(exception &e)
25     {
26         cout<<"异常:"<<e.what()<<endl;
27     }
28 }
```

运行结果：

```
e
e
hEllo string
hEllo sTring
异常:basic_string::at: __n (which is 1000) >= this->
size() (which is 12)
```

### 3、案例:字符串拼接2-1

```
1  /*
2  3.1.2.4 string拼接操作
3  string& operator+=(const string& str);//重载+=操作符
4  string& operator+=(const char* str);//重载+=操作符
5  string& operator+=(const char c);//重载+=操作符
6  string& append(const char *s);//把字符串s连接到当前字符串结尾
7  string& append(const char *s, int n);//把字符串s的前n个字符连接到当前字符串
  结尾
8  string& append(const string &s);//同operator+=()
9  string& append(const string &s, int pos, int n);//把字符串s中从pos开始的n个
  字符连接到当前字符串结尾
10 string& append(int n, char c);//在当前字符串结尾添加n个字符c
11 */
12 void test03()
13 {
14     string str1="hello";
15     string str2=" string";
16     //string& operator+=(const string& str);//重载+=操作符
17     str1 += str2;
18     cout<<str1<<endl;//"hello string"
19
20     string str3="hello";
21     //string& operator+=(const char* str);//重载+=操作符
22     str3 += " string";
23     cout<<str3<<endl;//"hello string"
24
25     string str4="hello";
26     //string& append(const char *s, int n);//把字符串s的前n个字符连接到当前字
  符串结尾
27     str4.append("hehehaha",4);
28     cout<<str4<<endl;//"hellohehe"
29
30     //string& append(const string &s, int pos, int n);//把字符串s中从pos开始
  的n个字符连接到当前字符串结尾
```

```

31 string str5="hello";
32 string str6="hehehahaha";
33 str5.append(str6,4,6);
34 cout<<str6<<endl;//"hellohahaha"
35 }

```

## 4、案例：字符串的查找替换

```

1  /*
2  3.1.2.5 string查找和替换
3  int find(const string& str, int pos = 0) const; //查找str第一次出现位置,从pos开始查找
4  int find(const char* s, int pos = 0) const; //查找s第一次出现位置,从pos开始查找
5  int find(const char* s, int pos, int n) const; //从pos位置查找s的前n个字符第一次位置
6  int find(const char c, int pos = 0) const; //查找字符c第一次出现位置
7  int rfind(const string& str, int pos = npos) const; //查找str最后一次位置,从pos开始查找
8  int rfind(const char* s, int pos = npos) const; //查找s最后一次出现位置,从pos开始查找
9  int rfind(const char* s, int pos, int n) const; //从pos查找s的前n个字符最后一次位置
10 int rfind(const char c, int pos = 0) const; //查找字符c最后一次出现位置
11 string& replace(int pos, int n, const string& str); //替换从pos开始n个字符为字符串str
12 string& replace(int pos, int n, const char* s); //替换从pos开始的n个字符为字符串s
13 */
14
15 #include<string.h>
16 void test04()
17 {
18     //int find(const string& str, int pos = 0) const; //查找str第一次出现位置,从pos开始查找
19     string str1="hehe:haha:xixi:haha:heihei";
20     //从str1中找haha
21     string tmp="haha";
22     cout<<str1.find(tmp)<<endl;//5
23     cout<<str1.find(tmp,10)<<endl;//15
24
25     //int find(const char* s, int pos = 0) const; //查找s第一次出现位置,从pos开始查找
26     cout<<str1.find("haha")<<endl;//5

```

```

27
28  str1.replace(5,4,"###");
29  cout<<str1<<endl;//"hehe:###:xixi:haha:heihei"
30
31  string str2="www.sex.117114.sex.person.77.com";
32  //需求：将字符串中的所有"sex"用***屏蔽
33  int ret = 0;
34  while((ret = str2.find("sex")) < str2.size())
35  {
36  str2.replace(ret,strlen("sex"),"***");
37  }
38  cout<<str2<<endl;
39  }

```

运行结果：

```

5
15
5
hehe:###:xixi:haha:heihei
www.***.117114.***.person.77.com

```

## 5、字符串比较

```

1  /*
2  3.1.2.6 string比较操作
3  compare函数在>时返回 1，<时返回 -1，==时返回 0。
4  比较区分大小写，比较时参考字典顺序，排越前面的越小。
5  大写的A比小写的a小。
6
7  int compare(const string &s) const;//与字符串s比较
8  int compare(const char *s) const;//与字符串s比较
9  */
10
11 void test05()
12 {
13  string str1="hehe";
14  string str2 = "haha";

```

```

15  cout<<str1.compare(str2)<<endl;//1
16  cout<<str1.compare("lala")<<endl;//-1
17  cout<<str1.compare("hehe")<<endl;//0
18  }

```

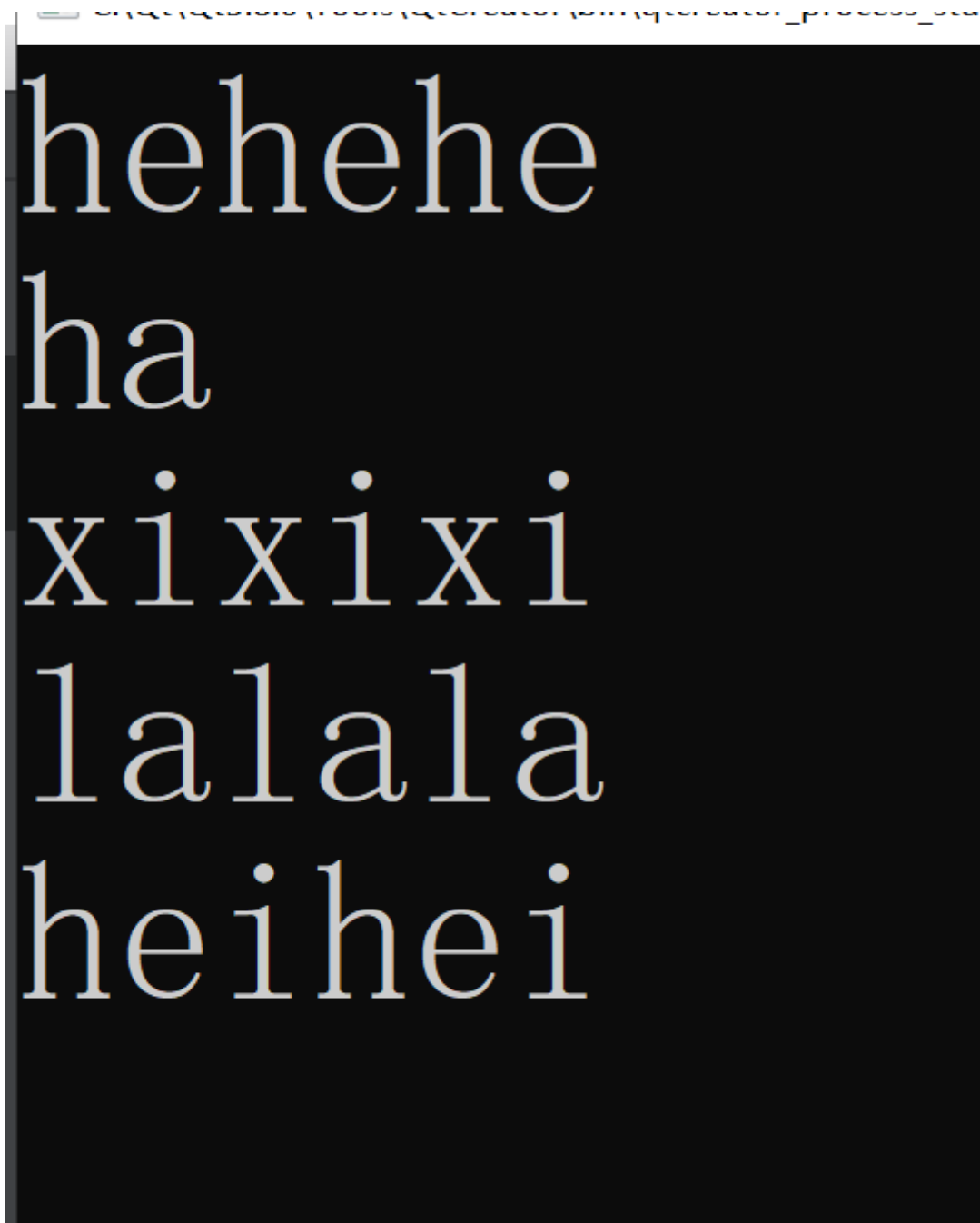
## 6、字符串提取

```

1  /*
2  3.1.2.7 string子串
3  string substr(int pos = 0, int n = npos) const; //返回由pos开始的n个字符组成的字符串
4  */
5  void test06()
6  {
7      string str1="hehehe:ha:xixixi:lalala:heihei";
8      //cout<<str1.substr(5,4)<<endl;
9
10     //案例:将:分割的所有字符串提取出来
11     int pos = 0;
12
13     while(1)
14     {
15         int ret = str1.find(":", pos);
16         if(ret < 0)
17         {
18             string tmp = str1.substr(pos, str1.size()-pos);
19             cout<<tmp<<endl;
20             break;
21         }
22
23         string tmp = str1.substr(pos, ret-pos);
24         cout<<tmp<<endl;
25
26         pos = ret+1;
27     }
28 }

```

运行结果：



## 7、字符串的插入删除2-2

```
1  /*
2  3.1.2.8 string插入和删除操作
3  string& insert(int pos, const char* s); //插入字符串
4  string& insert(int pos, const string& str); //插入字符串
5  string& insert(int pos, int n, char c); //在指定位置插入n个字符c
6  string& erase(int pos, int n = npos); //删除从Pos开始的n个字符
7  */
8
9  void test07()
10 {
11     string str1="hello world";
```

```

12  str1.insert(5,"hehe");
13  cout<<str1<<endl;//"hellohehe world
14
15  str1.erase(5,4);//删除字符串中hehe
16  cout<<str1<<endl;//"hello world"
17
18  //清空字符串 str1.size()得到字符串的总大小
19  str1.erase(0,str1.size());
20  cout<<str1.size()<<endl;//0
21  }

```

## 8、string 和c风格的字符串转换

```

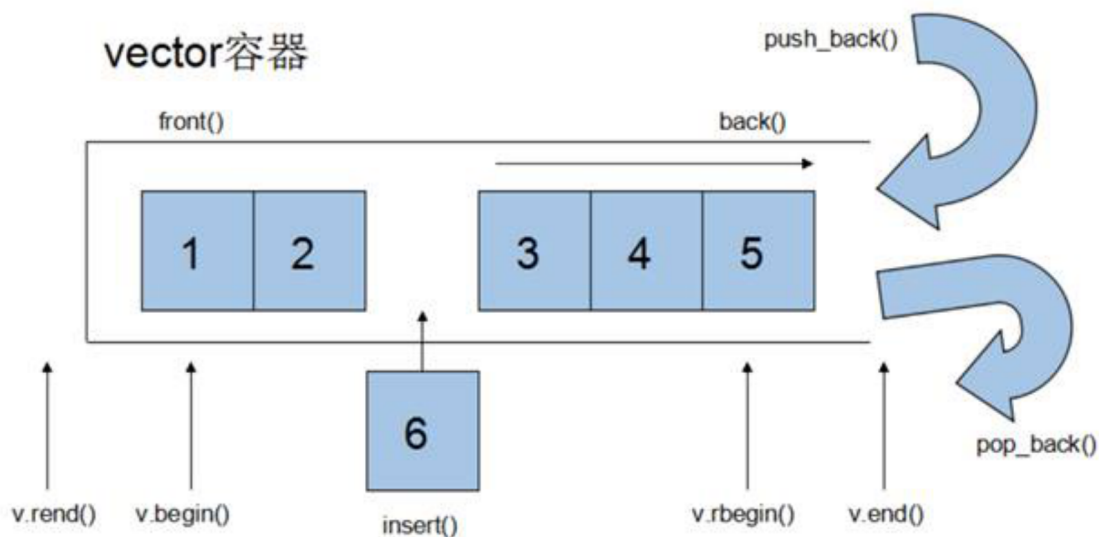
1  void test08()
2  {
3      string str1;//对象
4      char *str2 ="hello str";
5
6      //将char * 转成 string （直接完成）
7      str1 = str2;
8      cout<<str1<<endl;//hello str
9
10     string str3="hello str3";
11     //不能直接将string 转换成 char * 必须借助string中的c_str方法完成
12     //char *str4 = str3;//err
13     char *str4 = const_cast<char *> (str3.c_str());
14     cout<<str4<<endl;//"hello str3"
15 }

```

## 知识点4 【vector容器】单端动态数组

### 1、vector容器的概述





vector容器的迭代器：随机访问迭代器

随机访问迭代器：迭代器+n 可以通过编译 就是随机访问迭代器

## 2、vector的容量capacity和大小size的区别（了解）

capacity:空间能容纳元素最大个数。

size:空间中实际存放的元素个数。

```

4 void test01()
5 {
6     vector<int> v;
7     int i=0;
8     for(i=0;i<100;i++)
9         v.push_back(i);
10
11     cout<<"v的容量capacity:"<<v.capacity()<<endl;
12     cout<<"v的大小size:"<<v.size()<<endl;
13 }

```

C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator\_process\_stub.exe

v的容量capacity:128  
v的大小size:100

容量 >= 元素的个数size

为啥插入100个元素 容量确实128? 原因vector的未雨绸缪机制（后面会讲）

## 3、vector另寻地址的次数（了解）

```

1 void test02()
2 {
3     vector<int> v;
4     int *p=NULL;
5     int count = 0;//记录另寻地址的次数
6

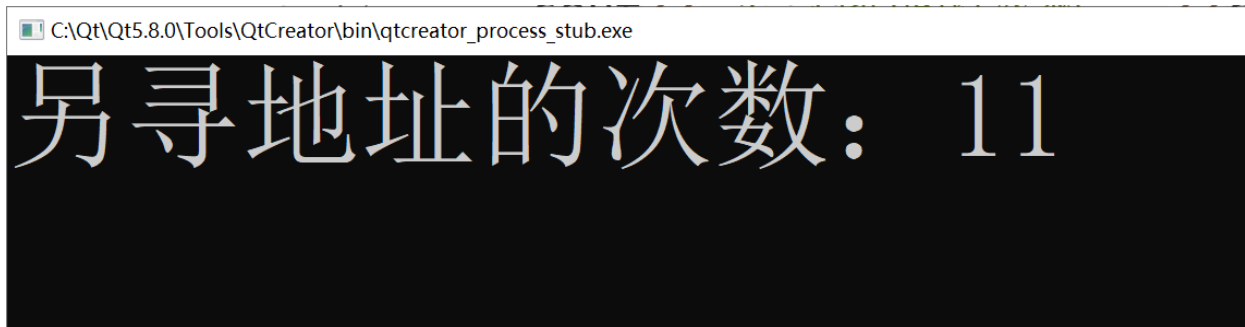
```

```

7   for(int i=0;i<1000;i++)
8   {
9       v.push_back(i);
10      if(p != &v[0])//如果p不等于&v[0] 说明另寻了新的空间
11      {
12          count++;
13          p = &v[0];
14      }
15  }
16
17  cout<<"另寻地址的次数: "<<count<<endl;
18  }

```

运行结果:



只要空间满 就会另寻空间。

#### 4、vector的未雨绸缪机制（了解）

```

1   void test03()
2   {
3       vector<int> v;
4       int *p = NULL;
5       int count =0;
6       for(int i=0;i<1000;i++)
7       {
8           if(p != &v[0])
9           {
10              cout<<"-----"<<count++<<"-----"<<endl;
11              p=&v[0];
12          }
13
14          v.push_back(i);
15          cout<<"cacapity = "<<v.capacity()<<"", size = "<<v.size()<<endl;
16      }
17  }

```

运行结果：

```
cacapity = 1, size = 1
-----0-----
cacapity = 2, size = 2
-----1-----
cacapity = 4, size = 3
-----2-----
cacapity = 4, size = 4
cacapity = 8, size = 5
-----3-----
cacapity = 8, size = 6
cacapity = 8, size = 7
cacapity = 8, size = 8
cacapity = 16, size = 9
-----4-----
cacapity = 16, size = 10
```

## 5、vector的构造函数

```
1
2  /*
3  3.2.4.1 vector构造函数
4  vector<T> v; //采用模板实现类实现，默认构造函数
5  vector(v.begin(), v.end()); //将v[begin(), end())区间中的元素拷贝给本身。
6  vector(n, elem); //构造函数将n个elem拷贝给本身。
7  vector(const vector &vec); //拷贝构造函数。
8  */
9  void printVectorInt(vector<int> &v)
10 {
11     for(vector<int>::iterator it=v.begin(); it!=v.end(); it++)
12     {
13         cout<<*it<<" ";
14     }
```

```
15  cout<<endl;
16  }
17  void test04()
18  {
19      //vector(n, elem);//构造函数将n个elem拷贝给本身
20      vector<int> v1(10,5);
21      printVectorInt(v1);
22
23      //vector(v.begin(), v.end());//将v[begin(), end())区间中的元素拷贝给本身
24      vector<int> v2(v1.begin()+2, v1.end()-2);
25      printVectorInt(v2);
26
27      vector<int> v3(v1);
28      printVectorInt(v3);
29  }
```

运行结果:

C:\Qt\Qt5.8.0\Tools\QtCreator\bin\qtcreator\_process\_stub.exe

```
5 5 5 5 5 5 5 5 5 5
5 5 5 5 5 5
5 5 5 5 5 5 5 5 5 5
```