初始vector

> vector 是一个 可以连续存储数据的容器，这里的数据可以是任意类型的，它的行为和数组相似， 访问vector中的元素和在末尾插入和删除都可以在常量级的时间复杂度情况下完成，而查找的特定值的元素所处的位置或者是在vector中插入元素则是线性时间复杂度，这里线性是指 O(n)

其实vector我们可以简单理解为一个线性表，或者是一个数组。在插入的时候我们要移动大量的数据，所以我们在选择他的时候要根据实际情况

- 在查找第几个数字，或者在末尾插入和删除数据使用次数比较多的情况下，可以选择vector
- 在vector中间插入和删除，给你数字让你查找该数字在第几个的时候，不建议使用

注：本文中提到的线性表和vector是一个意思 都是vector

vector常用的接口

一、构造函数

1、无参构造

```
vector<int> v;
```

注：vector其实可以理解为一个类，int是vector内存储数据的类型， 我们以整型为测试案例

2、有参构造

语法

```
vector( size_type num, const TYPE &val );
vector( const vector &from );
vector( input_iterator start, input_iterator end );
```

实例

```
    vector<int> v2(5, 0);
    vector<int> v3(v2);
    vector<int> v4(v3.begin(), v3.end());
```

介绍

1. 调用vector类里面的有参构造函数，意思是初始化数据 并赋值五个0
2. **又称为拷贝构造** ，利用v2构造v1
3. 利用迭代器进行构造v4；(一般用到迭代器均为左闭右开哦) v3.end() 其实是**指向线性表最后一个元素的下一个位置**

3、运算符重载函数

常用的运算符

```
v1 == v2
v1 != v2
v1 <= v2
v1 >= v2
v1 < v2
v1 > v2
v[]
```

介绍

这里介绍一下 == 的判断

判断 相等的条件有两个

1. 他们具有相等的容量
2. 他们相同位置的元素相等

其他的可以根据我后期文章《模拟实现vector来理解》

4、assign函数（赋值函数）

功能

对vector进行赋值

语法

```
void assign( input_iterator start, input_iterator end );
void assign( size_type num, const TYPE &val );
```

实例

```
    vector<int> v1(5, 0);
    v1.assign(4, 1);
    for(auto e : v1) cout << e << " ";

    vector<int> v2(5, 0);
    v2.assign(v2.begin(),v2.end());
    for (auto e : v2) cout << e << " ";
打印结果：1 1 1 1 0 0 0 0 0
```

介绍:

1. 利用assign函数对v1进行赋值，这里注意，他赋值的时候 会清空 **线性表原来的数据**
2. 利用assign传递迭代器进行赋值，

5、at函数

功能

以更安全的方式修改和拿到线性表中的数据

语法

```
TYPE at( size_type loc );
```

实例

```
    vector<int> v2(5, 0);
    for (int i = 0; i < v2.size(); i++)  v2[i] = i;
    cout << v2.at(2) << endl;
    v2.at(1) = 2;
    cout << v2.at(1) << endl;

打印结果: 2 2
```

介绍

其实这里的一般直接用[] 重载来代替 at函数也可以用来做赋值

其实没有[] 重载好使 ， 但是比【】更加安全

好处

1. 这段代码访问了vector末尾以后的元素,这将可能导致很危险的结果.以下的代码将更加安全:
2. 取代试图访问内存里非法值的作法,at() 函数能够辨别出访问是否越界并在越界的时候抛出一个异常

6、back 函数

功能

拿到线性表最后一个元素

语法

```
TYPE back();
```

实例

```
    vector<int> v1;
    for (int i = 0; i < 5; i++)
    {
        v1.push_back(i);
    }
    cout << v1. back() << endl;
打印结果为：4
```

介绍

这里就是拿到最后一个元素，作为函数的返回值，

7、begin 函数 和 end 函数

功能

拿到 线性表第一个元素的迭代器，和最后一个元素的下一个位置的迭代器

语法

```
iterator begin();
iterator end();
```

实例

```
    vector<int> v1 = { 0,1,2,3,4 };
    auto it = v1.begin();
    while (it != v1.end())
    {
        cout << *it << " ";
        ++it;
    }
打印结果为：0 1 2 3 4
```

介绍

这里返回类型是一个迭代器。我们暂时可以把迭代器理解为一个指针，比如begin函数返回的是指向第一个元素的指针。end是返回指向最后一个元素下一个元素的指针

8、capacity 函数 和 size 函数

功能

返回 当前线性表在不扩容的情况下最多能容纳的元素个数

返回 当前线性表所存储元素的个数

语法

```
 size_type capacity();
 size_type size();
```

实例

```
    vector<int> v1 = { 0,1,2,3,4,5 };
    v1.push_back(6);
    cout << v1.capacity() << endl;
    cout << v1.size() << endl;
打印结果为：  9
             7
```

9、clear 函数

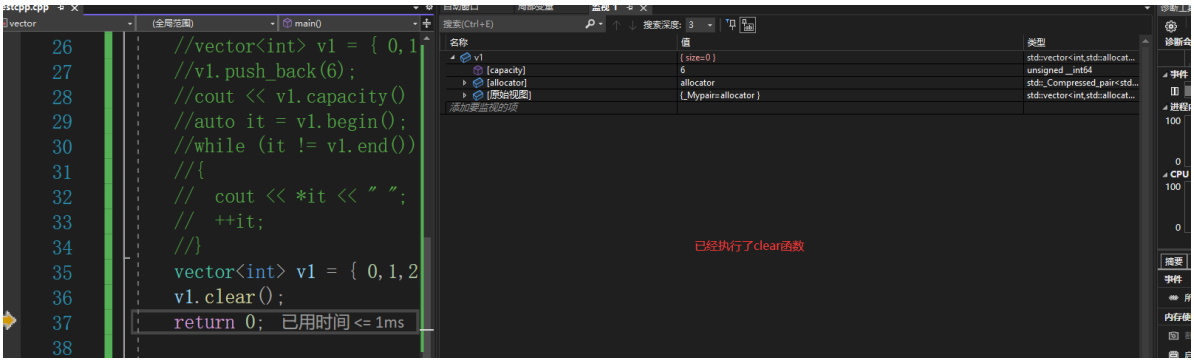
功能

删除当前线性表中所有的数据（不删除空间）

语法

```
void clear();
```

实例



介绍

我们可以从实例中看出来。我们不管有没有执行clear函数，capacity的值并没有改变。改变的是我们的数据多少个

10、empty 函数

功能

判断线性表是否为空表

语法

```
bool empty();
```

介绍

如果线性表为空表则返回true 如果线性表不为空则返回false

11、erase 函数

功能

删除线性表的某一个或者某个空间的数据

语法

```
iterator erase( iterator loc );
iterator erase( iterator start, iterator end );
```

实例

```
    vector<int> v1 = { 0,1,2,3,4,5 };
    cout << v1.size() << " ";
    for (auto e : v1) cout << e << " ";
    cout << endl;
    v1.erase(v1.begin() + 2);
    cout << v1.size() << " ";
    for (auto e : v1) cout << e << " ";
    cout << endl;

    v1.erase(v1.begin() + 1, v1.end());
    for (auto e : v1) cout << e << " ";
    cout << endl;
```

介绍

两段代码分开介绍

1. 我们可以发现 删除 数据其实把size也改变了
2. 我们可以通过这两段代码看出，他不仅可以用一个迭代器控制删除的某一个，还可以用两个迭代器控制删除的区间，左闭右开

12、insert 函数

功能

在任意位置插入元素

语法

```
  iterator insert( iterator loc, const TYPE &val );
  void insert( iterator loc, size_type num, const TYPE &val );
  void insert( iterator loc, input_iterator start, input_iterator end );
```

实例

```
    vector<int> v1 = { 0,1,2,3,4,5 };
    v1.insert(v1.begin() + 2, 5, 3);
    for (auto v : v1)   cout << v << " ";
    cout << endl;

    vector<int> v2 = { 0,1,2,3,4,5 };
    v2.insert(v2.begin() + 2, 3);
    for (auto v : v2) cout << v << " ";
    cout << endl;

    vector<int> v3 = { 0,1,2,3,4,5 };
    v3.insert(v3.begin(),v2.begin() + 1, v2.begin() + 4);
    v2.insert(v2.begin() + 2, 3);
    for (auto v : v3) cout << v << " ";
    cout << endl;
运行结果：  0 1 3 3 3 3 3 2 3 4 5
          0 1 3 2 3 4 5

          1 3 2 0 1 2 3 4 5
```

介绍

三种用法分开介绍

1. 在迭代器指向的位置前插入5个3
2. 在迭代器指向的位置前插入一个3
3. 在迭代器指向的位置前插入v2的迭代器区间表示的值

扩展

其实这里的inset可以与 算法 中的find 一起使用

例如：我们想在线性表指定数据前插入一个3；

代码：

```
    vector<int> v1 = { 0,1,2,3,4,5 };
    v1.insert(find(v1.begin(),v1.end(),2), 3);
    for (auto v : v1) cout << v << " ";
```

介绍一下find把

find 的用法一般为

find(star,end,val) 在迭代器 star和end中间查找第一个val

13、max_size 函数 没啥用

功能：就是打印最多能存储多少个数据

14、pop_back 和 push_back

功能

尾删和尾插

语法

```
void pop_back();
void push_back(const TYPE &val );
```

实例

```
    vector<int> v1 = { 0,1,2,3,4,5 };
    v1.push_back(6);
    for (auto v : v1) cout << v << " ";
    v1.pop_back();
运行结果：0 1 2 3 4 5 6
        0 1 2 3 4 5
```

15、rbegin 函数 和 rend函数

功能

返回指向当前线性表末尾的迭代器，和返回指向当前线性表第一个元素的前一个位置的迭代器

语法

```
  reverse_iterator rbegin();
  reverse_iterator rend();
```

实例

```
    vector<int> v1 = { 0,1,2,3,4,5 };
    auto it = v1.rbegin();
    while (it != v1.rend())
    {
        cout << *it << " ";
        it++;
    }
打印结果：5 4 3 2 1 0
```

介绍

倒序打印，如果你是顺着看的很容易理解意思的

16、reserve 函数 和 resize 函数

功能

均为改变size

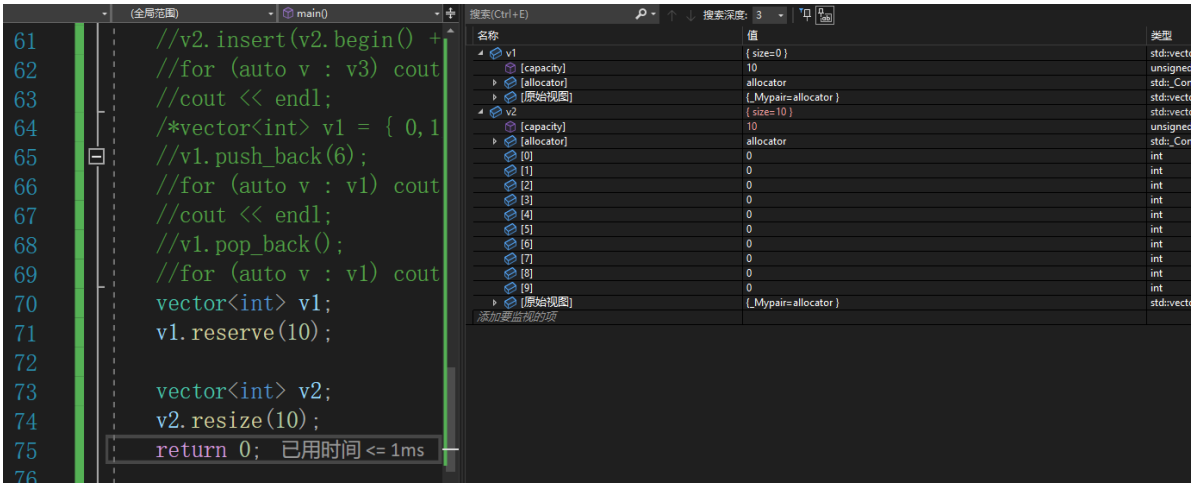
语法

```
  void reserve( size_type size );
  void resize( size_type size, TYPE val );
```

区别

reserve 函数为当前vector预留至少共容纳size个元素的空间

resize 函数改变当前vector的大小为size,且对新创建的元素赋值 为val 如果没有val就赋值为缺省值



17、swap函数

功能

交换两个线性表的元素

语法

```
void swap( vector &from );
```

实例

```
vector<int> v1 = { 0,1,2,3,4,5 };
vector<int> v2 = { 6,7,8,9 };
for (auto v : v1) cout << v << " ";
cout << endl;
for (auto v : v2) cout << v << " ";
cout << endl;

v1.swap(v2);

for (auto v : v1) cout << v << " ";
cout << endl;
for (auto v : v2) cout << v << " ";
cout << endl;
```