构造函数（（constructor））

接口说明:

①空向量

```
explicit vector(const A& al = A());
```

②创建n个value值的向量，第三个参数是具有默认值的空间配置对象

```
explicit vector(size_type n, const T& v = T(), const A& al = A());
```

③用vector对象拷贝构造对象

```
vector(const vector& x);
```

④用迭代器［begin, end）区间中的元素构造list，第三个参数是具有默认值的空间配置对象

```
vector(const_iterator first, const_iterator last,
       const A& al = A());
```

⑤用数组元素的地址做为参数创建链表

```
vector(_Iter _First, _Iter _Last)
            : _Mybase()
            {      // construct from [_First, _Last)
            _Construct(_First, _Last);
            }
```

①空向量

```
vector<int> v;
cout<<"size = "<<v.size()<<endl;  //0
cout<<"capacity = "<<v.capacity()<<endl;  //0
```

VS 编译器中 capacity容量是1.2倍增长

②创建向量n个value值，第三个参数是具有默认值的空间配置对象

```
vector<int> v(10, 2);
cout << "size = " << v.size() << endl;  //10
cout << "capacity = " << v.capacity() << endl;  //10
```

③用vector对象拷贝构造对象

```
vector<int> v1 = v;

cout << "size = " << v.size() << endl;  //10
cout << "capacity = " << v.capacity() << endl;  //10

cout << "size = " << v1.size() << endl;  //10
cout << "capacity = " << v1.capacity() << endl;  //
```

④用迭代器［begin，end）区间中的元素构造list，第三个参数是具有默认值的空间配置对象

```
vector<int> v(10, 2);
vector<int> v1(v.begin(), v.end());
```

⑤用数组元素的地址做为参数创建链表

```
int ar[] = {1,2,3,4,5,6,7,8,9,10};
vector<int> v1(ar, ar+sizeof(ar)/sizeof(int));

for(int i=0; i<v1.size(); ++i)  //1 2 3 4 5 6 7 8 9 10
   cout<<v1[i]<<" ";
cout<<endl;
```

遍历方式

1. 重载运算符[]

```
for(int i=0; i<v1.size(); ++i)
            cout<<v1[i]<<" ";
      cout<<endl;
```

2. at函数

```
for(int i=0; i<v1.size(); ++i)
            cout<<v1.at(i)<<" ";
      cout<<endl;
```

3. 利用迭代器

```
vector<int>::iterator it = v1.begin();
	while(it != v1.end())
	{
		cout<<*it<<" ";
		++it;
	}
	cout<<endl;
```

4. 反向迭代器

```
vector<int>::reverse_iterator rit = v1.rbegin();
	while(rit != v1.rend())
	{
		cout<<*rit<<" ";
		++rit;
	}
	cout<<endl;
```

5. auto变量

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

resize函数

```
vector<int> v(100,1);
cout<<"size = "<<v.size()<<endl;  //100
cout<<"capacity = "<<v.capacity()<<endl;  //100

v.resize(10, 2);
cout<<"size = "<<v.size()<<endl;  //10
cout<<"capacity = "<<v.capacity()<<endl;  //100

for(int i=0; i<v.size(); ++i)
cout<<v[i]<<" ";
cout<<endl;
```

size改变

capacity大才大

reserve函数

```
vector<int> v(10);
cout<<"size = "<<v.size()<<endl;  //10
cout<<"capacity = "<<v.capacity()<<endl;  //10

v.reserve(100);
cout<<"size = "<<v.size()<<endl;  //10
cout<<"capacity = "<<v.capacity()<<endl;  //100

v.reserve(50);
cout<<"size = "<<v.size()<<endl;  //10
cout<<"capacity = "<<v.capacity()<<endl;  //100
```

size不改变

capacity大才大

push_back函数

```
vector<int> v;
cout<<"size = "<<v.size()<<endl;
cout<<"capacity = "<<v.capacity()<<endl;

v.reserve(100);

for(int i=1; i<=100; ++i)
{
v.push_back(i);
cout<<i<<" size = "<<v.size()<<endl;
cout<<i<<" capacity = "<<v.capacity()<<endl;
}
```

find函数， insert函数 和 erase函数

```
int ar[] = {1,2,3,4,5,6,7,8,9,10};
int ar1[] = {11,22,33,44,55,66,77,88,99,100};
vector<int> v(ar, ar+10);
vector<int>::iterator pos;

pos = find(v.begin(), v.end(), 4);
v.insert(pos, 100);
pos = find(v.begin(), v.end(), 3);
v.erase(pos);   //erase  remove

for(int i=0; i<v.size(); ++i)  //1 2 100 4 5 6 7 8 9 10
    cout<<v[i]<<" ";
cout<<endl;
```

find第一二个参数是迭代器，表示在迭代器［第一个， 第二个 ）的范围类查找，最后一个参数是要查询的元素

swap函数

```
int ar[] = {1,2,3,4,5,6,7,8,9,10};
int ar1[] = {11,22,33,44,55,66,77,88,99,100};
vector<int> v(ar, ar+10);
vector<int> v1(ar1, ar1+10);

v.swap(v1);

for(int i=0; i<v.size(); ++i)  //11 22 33 44 55 66 77 88 99 100
    cout<<v[i]<<" ";
cout<<endl;

for(int i=0; i<v1.size(); ++i)  //1 2 3 4 5 6 7 8 9 10
    cout<<v1[i]<<" ";
cout<<endl;
```

交换两个vector对象的值