构造函数（ (constructor)）

接口说明

①explicit list(const A& al = A());

构造空的list

②explicit list(size_type n, const T& v = T(), const A& al = A());

构造的list中包含n个值为val的元素

③list (const list& x)

拷贝构造函数

④list(const_iterator first, const_iterator last, const A& al = A());

用迭代器 [first, last) 区间中的元素构造list

⑤list(_Iter _First, _Iter _Last);

用数组元素的地址做为参数创建链表

①

```
//list();
	list<int> L; //构造空的链表

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

②

```
//explicit list(size_type n, const T& v = T(), const A& al = A());
	list<int> L(10, 6); //构造的list中包含n个值为val的元素

	list<int>::iterator it = L.begin();
	while (it != L.end())
	{
		cout << *it << " ";
```

```
        ++it;
    }
    cout << endl;
```

③

```
//list(const list& x);
    list<int> L2(L);  //调用拷贝构造函数以对象构造对象
    it = L2.begin();
    while (it != L2.end())
    {
        cout << *it << " ";
        ++it;
    }
    cout << endl;
```

④

```
list<int>::iterator F = L.begin();
    list<int>::iterator T = F;
    int i = 3;
    while(i--)
        T++;

    //list(const_iterator first, const_iterator last,const A& al = A());
    list<int> L3(F, T);  //用迭代器 [first, tail) 区间中的元素构造list


    list<int>::iterator It = L3.begin();
    while (It != L3.end())
    {
        cout << *It << " ";
        ++It;
    }
    cout << endl;
```

⑤

```
int ar[10] = { 1,2,3,4,5,6,7,8,9,10 };
    list<int> mylist(ar, ar + 10);  //用数组元素的地址做为参数创建链表
    list<int>::iterator it = mylist.begin();
    while (it != mylist.end())
    {
        cout << *it << " ";
        ++it;
```

```
    }
    cout << endl;

    list<int>::reverse_iterator rit = mylist.rbegin();
    while (rit != mylist.rend())
    {
        cout << *rit << " ";
        ++rit; // --
    }
    cout << endl;
```

成员方法：

```
iterator begin();

const_iterator begin() const;
```

```
list<int> L;  //构造的list中包含n个值为val的元素
    L.push_back(1);
    L.push_back(2);

    list<int>::iterator it = L.begin();
    int arr[] = { 2, 3, 4, 5, 6, 7 };
    const list<int> L2(arr, arr+6);

    //L2.pop_back(5);  //const 限制的常链表不能修改
    //it = L2.end();  //const 限制的常链表不能用普通迭代器迭代

    //非const的迭代器不能指向const 限制的常链表
    //const list<int>::iterator conit = L2.end();  //error:-->
    const list<int>::const_iterator conit = L2.end();  //true

//不存在用户定义的从
//"std:List const iterator<std:: List val<std:conditional t<true, std:: List simple
types <int>
//, std:List iter types<int, size t, ptrdiff t, int*, const int*, int&, const int&,
std:List node<int, void*>*>>>>"到
//"std:List iterator<std:List val<std:conditional t<true, std:: List simple types <int>
//, std:List iter types<int, size t, ptrdiff t, int*, const int*, int&, const int&,
std:List node <int, void*>*>>>>"的适当转换

    const list<int>::iterator conit = L.end();
    list<int>::iterator it1 = L.end();
    *conit = 2;
    //*conit++;  //const的迭代器不能++（迭代器迭代的对象不能改变）
    *it1 = 2;
    *it1++;
```

const 限制的常链表不能用const或非const的迭代器迭代 ------> 错误

非const的迭代器不能指向const 限制的常链表 ------> 正确

const的迭代器不能++

```
iterator end();
iterator end() const;

reverse_iterator rbegin();
const_reverse_iterator rbegin() const;

reverse_iterator rend();
const_reverse_iterator rend() const;
```

```
void resize(size_type n, T x = T());
```

```
list<int> L;  //构造的list中包含n个值为val的元素
    L.push_back(1);
    L.push_back(2);
    cout<<L.size()<<endl;  //2

    L.resize(5);
    cout << L.size() << endl;  //5
```

```
size_type size() const;
```

```
list<int> L;
    L.push_back(1);
    L.push_back(2);
    cout << "L.size:    "<< L.size() << endl;
```

size的具体值就是链表真实结点的个数

```
    size_type max_size() const;
list<int> L;
cout << "L.max_size:"<< L.max_size() << endl;  //357913941
```

在VS2019平台下max_size数值是：357913941

```
    bool empty() const;
list<int> L;
    cout << "L.empty:    " << L.empty() << endl;  //返回值是1代表true

    A get_allocator() const;
    reference front();
    const_reference front() const;
    reference back();
    const_reference back() const;

    void push_front(const T& x);
    void pop_front();
    void push_back(const T& x);
    void pop_back();

    void assign(const_iterator first, const_iterator last);
    void assign(size_type n, const T& x = T());

 //void assign (size_type n, const value_type& val);
    first.assign(7, 100);                   // 7 ints with value 100
     for (auto e : first)
           cout << e << " ";
     cout << endl;
```

```
//void assign (InputIterator first, InputIterator last);
	second.assign(first.begin(), first.end()); // a copy of first
	for (auto e : second)
		cout << e << " ";
	cout << endl;

//assign(_Iter _First, _Iter _Last);
	int myints[] = { 1776,7,4 };
	first.assign(myints, myints + 3);
	for (auto e : first)
		cout << e << " ";
	cout << endl;
```

`void assign (size_type n, const value_type& val);` n个value值

`void assign (InputIterator first, InputIterator last);` 迭代器 [first, last) copy of first to last

`assign(_Iter _First, _Iter _Last);` 用数组的地址作为参数，将数组地址差中间的元素都插入

```
iterator insert(iterator it, const T& x = T());
void insert(iterator it, size_type n, const T& x);
void insert(iterator it,
    const_iterator first, const_iterator last);
void insert(iterator it,
    const T *first, const T *last);

iterator erase(iterator it);
iterator erase(iterator first, iterator last);

void clear();
void swap(list x);
void splice(iterator it, list& x);
void splice(iterator it, list& x, iterator first);
```

```
    void splice(iterator it, list& x, iterator first, iterator
last);
    void remove(const T& x);
    void remove_if(binder2nd<not_equal_to<T> > pr);s
    void unique();
    void unique(not_equal_to<T> pr);
    void merge(list& x);
    void merge(list& x, greater<T> pr);
    void sort();
    template<class Pred>
        void sort(greater<T> pr);
    void reverse();
```