C++ 容器类 <list>

C++ 标准库提供了丰富的功能，其中 <list> 是一个非常重要的容器类，用于存储元素集合，支持双向迭代器。<list> 是 C++ 标准模板库（STL）中的一个序列容器，它允许在容器的任意位置快速插入和删除元素。与数组或向量（<vector>）不同，<list> 不需要在创建时指定大小，并且可以在任何位置添加或删除元素，而不需要重新分配内存。

语法

以下是 <list> 容器的一些基本操作：

包含头文件：#include <list>

声明列表：std::list<T> mylist;，其中 T 是存储在列表中的元素类型。

插入元素：mylist.push_back(value);

删除元素：mylist.pop_back(); 或 mylist.erase(iterator);

访问元素：mylist.front(); 和 mylist.back();

遍历列表：使用迭代器 for (auto it = mylist.begin(); it != mylist.end(); ++it)

特点

双向迭代：<list> 提供了双向迭代器，可以向前和向后遍历元素。

动态大小：与数组不同，<list> 的大小可以动态变化，不需要预先分配固定大小的内存。

快速插入和删除：可以在列表的任何位置快速插入或删除元素，而不需要像向量那样移动大量元素。

声明与初始化

<list> 的声明和初始化与其他容器类似：

```
#include <iostream>
#include <list>

int main() {
    std::list<int> lst1;                  // 空的list
    std::list<int> lst2(5);               // 包含5个默认初始化元素的list
    std::list<int> lst3(5, 10);           // 包含5个元素，每个元素为10
    std::list<int> lst4 = {1, 2, 3, 4};   // 使用初始化列表

    return 0;
}
```

实例

下面是一个使用 <list> 的简单示例，包括创建列表、添加元素、遍历列表和输出结果。

实例

```cpp
#include <iostream>
#include <list>

int main() {
    // 创建一个整数类型的列表
    std::list<int> numbers;

    // 向列表中添加元素
    numbers.push_back(10);
    numbers.push_back(20);
    numbers.push_back(30);

    // 访问并打印列表的第一个元素
    std::cout << "First element: " << numbers.front() << std::endl;

    // 访问并打印列表的最后一个元素
    std::cout << "Last element: " << numbers.back() << std::endl;

    // 遍历列表并打印所有元素
    std::cout << "List elements: ";
    for (std::list<int>::iterator it = numbers.begin(); it != numbers.end(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;

    // 删除列表中的最后一个元素
    numbers.pop_back();

    // 再次遍历列表并打印所有元素
    std::cout << "List elements after removing the last element: ";
    for (std::list<int>::iterator it = numbers.begin(); it != numbers.end(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

输出结果:

```
First element: 10
Last element: 30
List elements: 10 20 30
List elements after removing the last element: 10 20
```

常用成员函数

以下是 <list> 中一些常用的成员函数:

函数	说明
push_back(const T& val)	在链表末尾添加元素
push_front(const T& val)	在链表头部添加元素
pop_back()	删除链表末尾的元素
pop_front()	删除链表头部的元素
insert(iterator pos, val)	在指定位置插入元素
erase(iterator pos)	删除指定位置的元素
clear()	清空所有元素
size()	返回链表中的元素数量
empty()	检查链表是否为空
front()	返回链表第一个元素
back()	返回链表最后一个元素
remove(const T& val)	删除所有等于指定值的元素
sort()	对链表中的元素进行排序
merge(list& other)	合并另一个已排序的链表
reverse()	反转链表
begin() / end()	返回链表的起始/结束迭代器

实例

1、基本操作

实例

```
#include <iostream>
#include <list>

int main() {
    std::list<int> lst = {10, 20, 30};

    // 插入和删除元素
    lst.push_front(5);           // 在头部插入5
    lst.push_back(40);           // 在尾部插入40
    lst.pop_front();             // 删除头部元素
    lst.pop_back();              // 删除尾部元素
```

```
    // 输出链表内容
    std::cout << "List elements: ";
    for (const auto& elem : lst) {
        std::cout << elem << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

2、插入和删除特定位置的元素

实例

```
#include <iostream>
#include <list>

int main() {
    std::list<int> lst = {1, 2, 3, 4, 5};
    auto it = lst.begin();
    std::advance(it, 2);          // 移动迭代器到第3个元素（值为3）

    lst.insert(it, 10);           // 在第3个元素前插入10
    lst.erase(it);                // 删除第3个元素

    // 输出链表内容
    std::cout << "List elements: ";
    for (const auto& elem : lst) {
        std::cout << elem << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

3、排序和去重

实例

```
#include <iostream>
#include <list>

int main() {
    std::list<int> lst = {3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5};
    lst.sort();                   // 排序
    lst.unique();                 // 删除相邻重复元素

    // 输出链表内容
    std::cout << "Sorted and unique list: ";
    for (const auto& elem : lst) {
        std::cout << elem << " ";
    }
    std::cout << std::endl;
```

```
    return 0;
}
```

4、合并和反转

实例

```
#include <iostream>
#include <list>

int main() {
    std::list<int> lst1 = {1, 3, 5, 7};
    std::list<int> lst2 = {2, 4, 6, 8};

    lst1.merge(lst2);              // 合并两个已排序的链表
    lst1.reverse();                // 反转链表

    // 输出链表内容
    std::cout << "Merged and reversed list: ";
    for (const auto& elem : lst1) {
        std::cout << elem << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

与其他容器对比

特性	std::list	std::vector	std::deque
内存结构	非连续内存，双向链表	连续内存	分段连续内存
访问性能	顺序访问较快，随机访问慢	随机访问快	末尾和头部访问都快
插入/删除性能	任意位置插入、删除快	末尾插入快，中间位置慢	头尾插入、删除快
适用场景	频繁在中间插入/删除	需要高效随机访问	需要在头尾快速插入/删除
迭代器稳定性	稳定，元素插入或删除不会失效	插入、删除可能导致迭代器失效	插入、删除可能导致迭代器失效

注意事项

<list> 的元素是按插入顺序存储的，而不是按元素值排序。

由于 <list> 的元素存储在不同的内存位置，所以它不适合需要随机访问的场景。

与向量相比，<list> 的内存使用效率较低，因为每个元素都需要额外的空间来存储指向前后元素的指针。

通过这个简单的介绍和示例，初学者应该能够对 C++ 的 <list> 容器有一个基本的了解，并能够开始使用它来解决实际问题。