STL容器

2024年6月23日 18:35

三、STL容器

3.1 STL容器初识

3.2 string容器

3.3 vector容器

3.4 depue容器

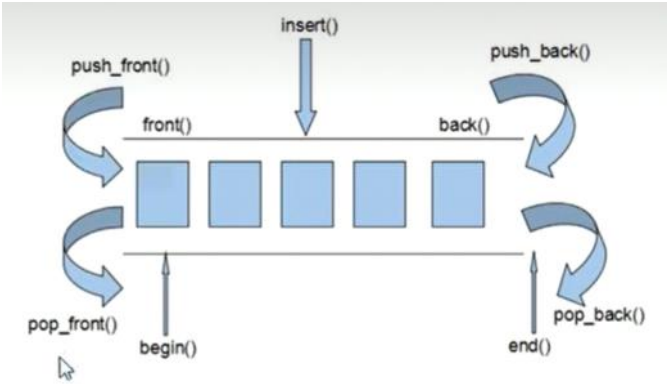
3.4.1 deque的基本概念

双端数组，可以对头部进行插入和删除等操作

1、deque与vector区别：

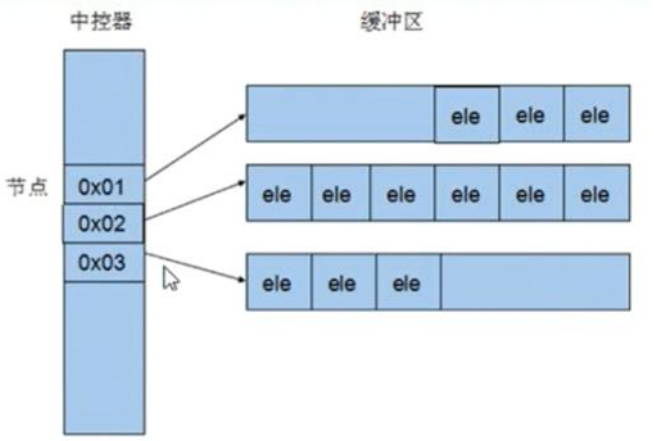
- vector对于头部的插入删除效率低，数据量越大，效率越低
- deque相对而言，对头部的插入删除速度回比vector快
- vector访问元素时的速度会比deque快，这和两者内部实现有关

vector 在内存中是一块连续的线性空间，而depue不同。deque内部有个中控器，维护每段缓冲区中的内容



2、deque内部工作原理：

deque内部有个中控器，维护每段缓冲区中的内容，缓冲区中存放真实数据中控器维护的是每个缓冲区的地址，使得使用deque时像一片连续的内存空间。



3.4.2 deque构造函数

```
deque<T> deqT;          //默认构造形式
deque(beg, end);        //构造函数将[beg,end)区间中的元素拷贝给本身,
deque(n, elem);         //构造函数将n个elem拷贝给本身。
deque(const deque &deg);//拷贝构造函数
```

3.4.3 deque赋值

函数原型:

```
deque& operator(const deque &deq); //重载等号操作符
assign(beg, end);                  //将[beg, end)区间中的数据拷贝赋值给本身,
assign(n, elem);                   //将n个elem拷贝赋值给本身。
```

3.4.4 deque大小操作

```
deque.empty();          //判断容器是否为空
deque.size();           //返回容器中元素的个数
deque.resize(num);      //重新指定容器的长度为num,若容器变长，则以默认值填充新位置。
                        //如果容器变短，则末尾超出容器长度的元素被删除。
deque.resize(num,elem); //重新指定容器的长度为num,若容器变长，则以elem值填充新位置。
                        //如果容器变短，则末尾超出容器长度的元素被删除。
```

3.4.5 deque插入删除

两端插入操作:

```
push_back(elem);    //尾插，在容器尾部添加一个数据
push_front(elem);   //头插，在容器头部插入一个数据
pop_back();         //尾删，删除容器最后一个数据
pop_front();        //头删，删除容器第一个数据
```

指定位置操作:

```
insert(pos,elem);    //在pos位置插入一个elem元素的拷贝，返回新数据的位置。
insert(pos,n,elem);  //在pos位置插入n个elem数据，无返回值。
insert(pos,beg,end); //在pos位置插入[beg,end)区间的数据，无返回值。
clear();             //清空容器的所有数据
erase(beg,end);      //删除[beg,end)区间的数据，返回下一个数据的位置。
erase(pos);          //删除pos位置的数据，返回下一个数据的位置。
```

3.4.5 deque数据存取

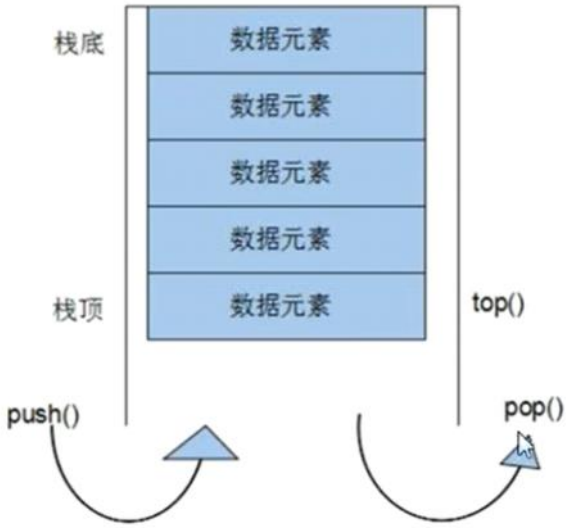
```
at(int idx);   //返回索引idx所指的数据
```

```
operator[];     //返回索引idx所指的数据
front();        //返回容器中第一个数据元素
back();         //返回容器中最后一个数据元素
```

3.5 stack-栈容器

3.5.1 stack基本概念

概念:stack是一种先进后出(First In Last Out,FILO)的数据结构，它只有一个出口



栈中只有顶端的元素才可以被外界使用，因此栈不允许有遍历行为

3.5.2 stack的接口

功能描述:栈容器常用的对外接口

构造函数:

```
stack<T> stk;   // stack采用模板类实现，stack对象的默认构造形式
stack(const stack &stk);  // 拷贝构造函数
```

赋值操作:

```
stack& operator=(const stack &stk);   // 重载等号操作符
```

数据存取:

```
push(elem);  // 向栈顶添加元素
pop();     // 从栈顶移除第一个元素
top();     // 返回栈顶元素
```

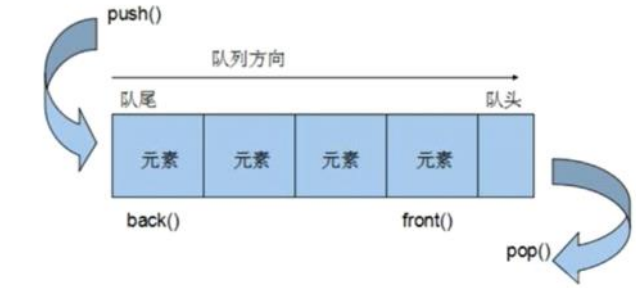
大小操作:

```
empty(); // 栈是否为空
size();  // 返回栈的大小
```

3.6 queue-队列

3.6.1 queue基本概念

概念:Queue是一种先进先出(First In First Out,FIFO)的数据结构，它有两个出口



队尾进，队头出

3.6.2 queue的接口

构造函数:

```
queue<T> que;             //queue采用模板类实现，queue对象的默认构造形式
queue(const queue &que); //拷贝构造函数
```

赋值操作:

```
queue& operator=(const queue &que);  //重载等号操作符
```

数据存取:

```
push(elem); //往队尾添加元素
pop();      //从队头移除第一个元素
back();     //返回最后一个元素
front();    //返回第一个元素
```

大小操作:

```
empty();   //判断堆栈是否为空
size();    //返回栈的大小
```

3.7 list-链表

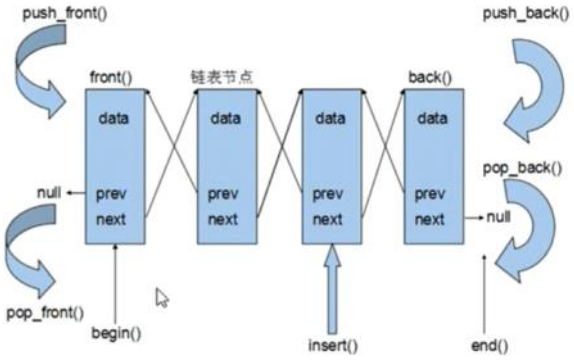
3.7.1 list基本概念

功能:将数据进行链式存储

链表(list)是一种物理存储单元上非连续的存储结构，数据元素的逻辑顺序是通过链表中的指针链接实现的链表的组成:链表由一系列结点组成

结点的组成:一个是存储数据元素的数据域，另一个是存储下一个结点地址的指针域

STL中的链表是一个双向循环链表



由于链表的存储方式并不是连续的内存空间，因此链表list中的迭代器只支持前移和后移，属于双向迭代器

list的优点:

- 采用动态存储分配，不会造成内存浪费和溢出
- 链表执行插入和删除操作十分方便，修改指针即可，不需要移动大量元素

list的缺点:

- 链表灵活，但是空间(指针域)和 时间(遍历)额外耗费较大

List有一个重要的性质，插入操作和删除操作都不会造成原有list迭代器的失效，这在vector是不成立的。

总结:STL中List和vector是两个最常被使用的容器，各有优缺点

3.7.2 构造函数

```
list<T> lst; //list采用采用模板类实现,对象的默认构造形式!
list(beg,end); //构造函数将[beg, end)区间中的元素拷贝给本身。
list(n,elem);  //构造函数将n个elem拷贝给本身。
list(const list &lst); //拷贝构造函数。
```

3.7.3 赋值与交换

```
assign(beg,end); //将[beg,end)区间中的数据拷贝赋值给本身
assign(n, elem); //将n个elem拷贝赋值给本身
list& operator=(const list &lst); //重载等号操作符
swap(lst);//将lst与本身的元素互换。
```

3.7.4 大小操作

```
size();  //返回容器中元素的个数
empty(); //判断容器是否为空
resize(num); //重新指定容器的长度为num，若容器变长，则以默认值填充新位置。
             //如果容器变短，则末尾超出容器长度的元素被删除。
resize(num,elem);  //重新指定容器的长度为num，若容器变长，则以elem值填充新位置。
                   //如果容器变短，则末尾超出容器长度的元素被删除。
```

3.7.5 插入删除

```
push_back(elem);//在容器尾部加入一个元素
pop_back();//删除容器中最后一个元素
push_front(elem);//在容器开头插入一个元素
pop_front();//从容器开头移除第一个元素
insert(pos,elem);//在pos位置插elem元素的拷贝，返回新数据的位置
insert(pos,n,elem);//在pos位置插入n个elem数据，无返回值。
insert(pos,beg,end);//在pos位置插入[beg,end)区间的数据，无返回值。
```

```
clear();//移除容器的所有数据
erase(beg,end);//删除[beg,end)区间的数据，返回下一个数据的位置,
erase(pos);//删除pos位置的数据，返回下一个数据的位置。
remove(elem);//删除容器中所有与elem值匹配的元素。
```

3.7.6 数据存取

```
front(); // 返回第一个元素。
back();  // 返回最后一个元素
```

3.7.7 反转和排序

```
reverse();  //反转链表
sort();     //链表排序
```

3.8 set/multiset容器（集合容器）

3.8.1 set/multiset基本概念

简介:

所有元素都会在插入时自动被排序

本质:

get/multiset属于关联式容器，底层结构是用二叉树实现。

set和multiset区别:

- set不允许容器中有重复的元素
- multiset允许容器中有重复的元素

3.8.2 set/multiset 构造赋值

构造:

```
set<T> st;          //默认构造函数:
set(const set &st); //拷贝构造函数
```

赋值:

```
set& operator=(const set &st);  //重载等号操作符
```

3.8.3 set/multiset 大小和交换

```
size(); //返回容器中元素的数目
empty(); //判断容器是否为空
swap(st); //交换两个集合容器
```

3.8.4 set/multiset 插入和删除

```
insert(elem);    //在容器中插入元素。
clear();         //清除所有元素
erase(pos);      //删除pos 迭代器所指的元素，返回下一个元素的迭代器。
erase(beg, end); //删除区间[beg,end)的所有元素，返回下一个元素的迭代器
erase(elem);    //删除容器中值为elem的元素。
```

3.8.5 set/multiset 查找和统计

```
find(key); //查找key是否存在,若存在，返回该键的元素的迭代器;若不存在，返回set.end();
```

count(key); //统计key的元素个数

3.8.6 set和multiset区别

set不可以插入重复数据，而multiset可以
set插入数据的同时会返回插入结果，表示插入是否成功
multiset不会检测数据，因此可以插入重复数据

set 插入时，会返回一个pair的容器

pair，对组，有两个值，一个是插入的位置，一个bool是是否插入成功

```
void test02()
{
    set<int> st;
    // set的插入会返回一个队组。pair<iterator, bool>
    // 插入位置的迭代器和是否插入成功
    pair<set<int>::iterator, bool>  ret = st.insert(8);
    if (ret.second)
    {
        cout << "第一次插入成功！" << endl;
    }
    else
    {
        cout << "第一次插入失败！" << endl;
    }
    ret = st.insert(8);
    if (ret.second)
    {
        cout << "第二次插入成功！" << endl;
    }
    else
    {
        cout << "第二次插入失败！" << endl;
    }
    printSet(st);
}
```

3.8.7 对组的创建

功能描述:
成对出现的数据，利用对组可以返回两个数据
两种创建方式:

```
pair<type,type>p( value1,value2 );
pair<type,type>p=make_pair( value1,value2 );
```

3.8.8 set排序

学习目标:
set容器默认排序规则为从小到大，掌握如何改变排序规则
主要技术点:
利用仿函数，可以改变排序规则

仿函数：重载了函数调用的小括号 ()

```
// set 存放内置类型的排序
class MyCmp
{
public:
    // () 操作符的重载
    // const 限定 调用 () 的set对象不能修改v1, v2的值
    bool operator()(int v1,int v2) const
    {
        return v1 > v2;
    }
};
void test04()
{
    // 默认从小到大
    set<int> st;
    st.insert(5);
    st.insert(2);
    st.insert(1);
    st.insert(3);
    st.insert(4);

    printSet(st);

    // 指定排序规则为 从大到小
    set<int, MyCmp> s2;
    s2.insert(5);
    s2.insert(2);
    s2.insert(1);
    s2.insert(3);
    s2.insert(4);

    // 迭代器也要改变
    for (set<int, MyCmp>::iterator it = s2.begin(); it != s2.end(); it++)
    {
        cout << *it << " ";
    }
    cout << endl;

}


// set 存放自定义数据类型的排序
class Person
{
public:
    Person(string name, int age)
    {
        this->m_Name = name;
        this->m_Age = age;
    }

    string m_Name;
    int m_Age;
};

class MyCmpByAge
{
public:
    bool operator()(const Person& p1, const Person& p2) const
    {
```

```
        return p1.m_Age > p2.m_Age;
    }
};
void test05()
{
    // 自定义的数据类型，都会指定排序规则
    set<Person, MyCmpByAge> st;

    Person p1("张三", 35);
    Person p2("李四", 36);
    Person p3("王五", 32);
    Person p4("赵六", 33);
    Person p5("钱七", 38);

    st.insert(p1);
    st.insert(p2);
    st.insert(p3);
    st.insert(p4);
    st.insert(p5);

    for (set<Person, MyCmpByAge>::iterator it = st.begin(); it != st.end(); it++)
    {
        cout << "姓名：" << it->m_Name << ", 年龄：" << it->m_Age << endl;
    }

}
```

3.9 map/multimap容器（集合容器）

3.9.1 基本概念

简介:

- map中所有元素都是pair
- pair中第一个元素为key(键值)，起到索引作用，第二个元素为value(实值)
- 所有元素都会根据元素的键值自动排序

本质:

- map/multimap属于关联式容器，底层结构是用二叉树实现。

优点:

- 可以根据key值快速找到value值

map和multimap区别:

- map不允许容器中有重复key值元素
- multimap允许容器中有重复key值元素

3.9.2 map构造和赋值

功能描述:

- 对map容器进行构造和赋值操作

构造:

```
map<T1,T2> mp;    //map默认构造函数:
map(const maÃ&mp);//拷贝构造函数
```

赋值:

```
map& operator=(const map &mp); //重载等号操作符
```

3.9.3 map大小和交换

功能描述:

统计map容器大小以及交换map容器

函数原型:

```
size();   //返回容器中元素的数目
empty();  //判断容器是否为空
swap(st); //交换两个集合容器
```

3.9.4 map插入和删除

功能描述:

map容器进行插入数据和删除数据

函数原型:

```
insert(elem);    //在容器中插入元素。
clear();         //清除所有元素
erase(pos);      //删除pos迭代器所指的元素，返回下一个元素的迭代器,
erase(beg, end); //删除区间[beg,end)的所有元素，返回下一个元素的迭代器。
erase(key);      //删除容器中值为key的元素。
```

四种插入方式

```
void test01()
{
    map<int, string> mp;

    //插入  4种
    mp.insert(pair<int, string>(1, "张三"));
    mp.insert(make_pair(2,"李四"));
    mp.insert(map<int, string>::value_type(3, "王五"));
    mp[4] = "赵六"; // 不建议用来插入

    //  mp[5]; // key不存在时，会创建一个 key为5，value为默认值的值

    // 如果确认key存在，可以根据key找的 key的value值
    cout << mp[4] << endl; // 赵六

    printMap(mp);
}
```

3.9.5 map查找和统计

功能描述:

对map容器进行查找数据以及统计数据

函数原型:

```
find(key);  //查找key是否存在,若存在，返回该键的元素的迭代器;若不存在，返回set.end();
count(key); //统计key的元素个数
```

3.9.6 map容器排序

学习目标:

map容器默训排序规则为 按照key值进行 从小到大排序，掌握如何改变排序规则

主要技术点:

利用仿函数，可以改变排序规则

```
// map容器排序
// map容器默训排序规则为 按照key值进行 从小到大排序
// 改为从大到小
// 内置数据类型
class MyCmp
{
public:
    bool operator()(int v1,int v2) const
```

```
    {
        return v1 > v2;
    }
};
void test02()
{
    map<int, string,MyCmp> mp;

    // 插入
    mp.insert(pair<int, string>(3, "张三"));
    mp.insert(make_pair(1, "李四"));
    mp.insert(map<int, string>::value_type(2, "王五"));
    mp[4] = "赵六"; // 不建议用来插入

    for (map<int, string, MyCmp>::iterator it = mp.begin(); it != mp.end(); it++)
    {
        cout << "编号：" << it->first << ", 姓名：" << it->second << endl;
    }
}

//  自定义数据类型
class Person
{
public:
    Person()
    {

    }
    Person(string name, int age)
    {
        this->m_Name = name;
        this->m_Age = age;
    }

    string m_Name;
    int m_Age;
};

class MyCmp1
{
public:
    bool operator()(int v1, int v2) const
    {
        return v1 > v2;
    }
};

void test03()
{
    map<int, Person, MyCmp1> mp;

    Person p1("张三", 34);
    Person p2("李四", 28);
    Person p3("王五", 53);
    Person p4("赵六", 23);

    // 插入
    mp.insert(pair<int, Person>(3, p1));
    mp.insert(make_pair(1, p2));
    mp.insert(map<int, Person, MyCmp1>::value_type(2, p3));
    mp[4] = p4; // 不建议用来插入
```

```
    for (map<int, Person, MyCmp1>::iterator it = mp.begin(); it != mp.end(); it++)
    {
        cout << "编号：" << it->first << ", 姓名：" << it->second.m_Name << ", 年龄："<< it->
        second.m_Age << endl;
    }
}
```