2.2 STL基本概念

- STL(Standard Template Library,**标准模板库**)
- STL 从广义上分为: **容器(container) 算法(algorithm) 迭代器(iterator)**
- **容器**和**算法**之间通过**迭代器**进行无缝连接。
- STL 几乎所有的代码都采用了模板类或者模板函数

2.3 STL六大组件

STL大体分为六大组件，分别是:**容器、算法、迭代器、仿函数、适配器（配接器）、空间配置器**

STL**容器**就是将运用**最广泛的一些数据结构**实现出来

常用的数据结构：数组, 链表,树, 栈, 队列, 集合, 映射表 等

这些容器分为**序列式容器**和**关联式容器**两种:

序列式容器:强调值的排序，序列式容器中的每个元素均有固定的位置。
关联式容器:二叉树结构，各元素之间没有严格的物理上的顺序关系

种类	功能	支持运算
输入迭代器	对数据的只读访问	只读，支持++、==、！=
输出迭代器	对数据的只写访问	只写，支持++
前向迭代器	读写操作，并能向前推进迭代器	读写，支持++、==、！=
双向迭代器	读写操作，并能向前和向后操作	读写，支持++、–,
随机访问迭代器	读写操作，可以以跳跃的方式访问任意数据，功能最强的迭代器	读写，支持++、–、[n]、-n、<、<=、>、>=

容器： `vector`

算法： `for_each`

迭代器： `vector<int>::iterator`

- char * 是一个指针
- string是一个类，类内部封装了char*，管理这个字符串，是一个char*型的容器。

构造函数原型：

- `string();` //创建一个空的字符串 例如: string str;
  `string(const char* s);` //使用字符串s初始化
- `string(const string& str);` //使用一个string对象初始化另一个string对象
- `string(int n, char c);` //使用n个字符c初始化

3.1.3 string赋值操作

功能描述：

- 给string字符串进行赋值

赋值的函数原型：

- `string& operator=(const char* s);` //char*类型字符串 赋值给当前的字符串
- `string& operator=(const string &s);` //把字符串s赋给当前的字符串
- `string& operator=(char c);` //字符赋值给当前的字符串
- `string& assign(const char *s);` //把字符串s赋给当前的字符串
- `string& assign(const char *s, int n);` //把字符串s的前n个字符赋给当前的字符串
- `string& assign(const string &s);` //把字符串s赋给当前字符串
- `string& assign(int n, char c);` //用n个字符c赋给当前字符串

字符串拼接

函数原型：

- `string& operator+=(const char* str);` //重载+=操作符
- `string& operator+=(const char c);` //重载+=操作符
- `string& operator+=(const string& str);` //重载+=操作符
- `string& append(const char *s);` //把字符串s连接到当前字符串结尾
- `string& append(const char *s, int n);` //把字符串s的前n个字符连接到当前字符串结尾
- `string& append(const string &s);` //同operator+=(const string& str)
- `string& append(const string &s, int pos, int n);` //字符串s中从pos开始的n个字符连接到字符串结尾

查找和替换

函数原型：

- `int find(const string& str, int pos = 0) const;` //查找str第一次出现位置,从pos开始查找
- `int find(const char* s, int pos = 0) const;` //查找s第一次出现位置,从pos开始查找
- `int find(const char* s, int pos, int n) const;` //从pos位置查找s的前n个字符第一次位置
- `int find(const char c, int pos = 0) const;` //查找字符c第一次出现位置
- `int rfind(const string& str, int pos = npos) const;` //查找str最后一次位置,从pos开始查找
- `int rfind(const char* s, int pos = npos) const;` //查找s最后一次出现位置,从pos开始查找
- `int rfind(const char* s, int pos, int n) const;` //从pos查找s的前n个字符最后一次位置
- `int rfind(const char c, int pos = 0) const;` //查找字符c最后一次出现位置
- `string& replace(int pos, int n, const string& str);` //替换从pos开始n个字符为字符串str
- `string& replace(int pos, int n,const char* s);` //替换从pos开始的n个字符为字符串s

比较

函数原型：

- `int compare(const string &s) const;` //与字符串s比较
- `int compare(const char *s) const;` //与字符串s比较

3.1.7 string字符存取

string中单个字符存取方式有两种

- `char& operator[](int n);` //通过[]方式取字符
- `char& at(int n);` //通过at方法获取字符

插入和删除

函数原型：

- `string& insert(int pos, const char* s);` //插入字符串
- `string& insert(int pos, const string& str);` //插入字符串
- `string& insert(int pos, int n, char c);` //在指定位置插入n个字符c
- `string& erase(int pos, int n = npos);` //删除从Pos开始的n个字符

3.1.9 string子串

功能描述：

- 从字符串中获取想要的子串

函数原型：

- `string substr(int pos = 0, int n = npos) const;` //返回由pos开始的n个字符组成的字符串

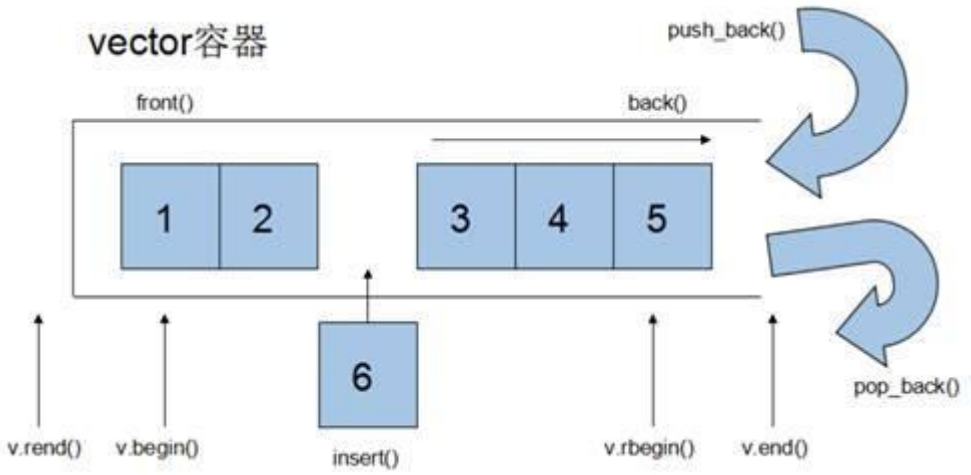
3.2.1 vector基本概念

功能：

- vector数据结构和**数组非常相似**，也称为**单端数组**

vector与普通数组区别：

- 不同之处在于数组是静态空间，而vector可以**动态扩展**



函数原型：

- `vector<T> v;` //采用模板实现类实现，默认构造函数
- `vector(v.begin(), v.end());` //将v[begin(), end())区间中的元素拷贝给本身。
- `vector(n, elem);` //构造函数将n个elem拷贝给本身。
- `vector(const vector &vec);` //拷贝构造函数。

- 给vector容器进行赋值

函数原型：

- `vector& operator=(const vector &vec);` //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。

功能描述：

- 对vector容器的容量和大小操作

函数原型：

- `empty();` //判断容器是否为空
- `capacity();` //容器的容量
- `size();` //返回容器中元素的个数
- `resize(int 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(int 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

> - 判断是否为空 — empty
> - 返回元素个数 — size
> - 返回容器容量 — capacity
> - 重新指定大小 — resize

- 对vector容器进行插入、删除操作

函数原型：

- `push_back(ele);` //尾部插入元素ele
- `pop_back();` //删除最后一个元素
- `insert(const_iterator pos, ele);` //迭代器指向位置pos插入元素ele
- `insert(const_iterator pos, int count,ele);` //迭代器指向位置pos插入count个元素ele
- `erase(const_iterator pos);` //删除迭代器指向的元素
- `erase(const_iterator start, const_iterator end);` //删除迭代器从start到end之间的元素
- `clear();` //删除容器中所有元素

- 对vector中的数据的存取操作

函数原型：

- `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

- 实现两个容器内元素进行互换

函数原型：

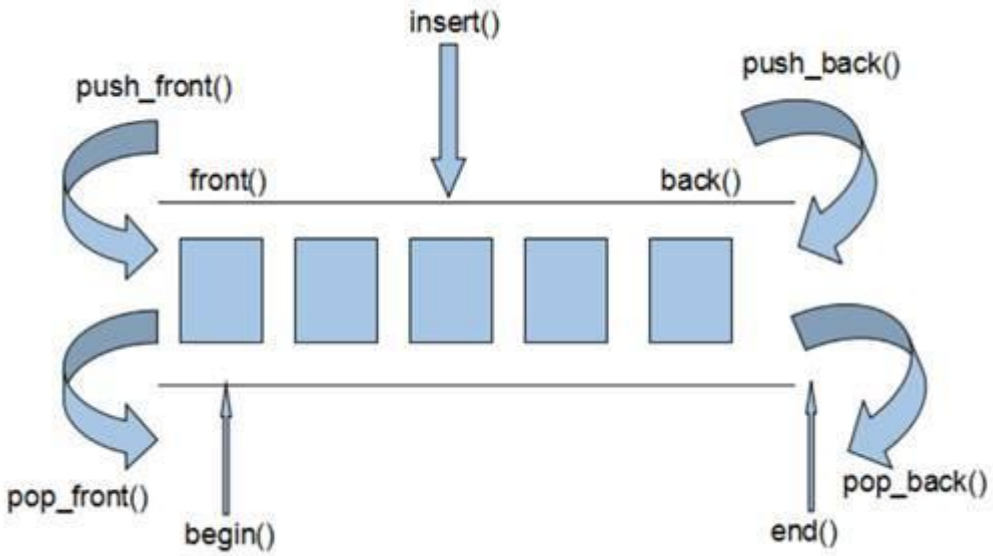
- `swap(vec);` // 将vec与本身的元素互换

- 减少vector在动态扩展容量时的扩展次数

函数原型：

- `reserve(int len);` //容器预留len个元素长度，预留位置不初始化，元素不可访问。

deque



- deque容器构造

函数原型：

- `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 &deq);` //拷贝构造函数

- 给deque容器进行赋值

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。

- 对deque容器的大小进行操作

函数原型：

- `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。

- 向deque容器中插入和删除数据

函数原型：

两端插入操作：

- `push_back(elem);` //在容器尾部添加一个数据
- `push_front(elem);` //在容器头部插入一个数据
- `pop_back();` //删除容器最后一个数据
- `pop_front();` //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。

- 对deque 中的数据的存取操作

函数原型：

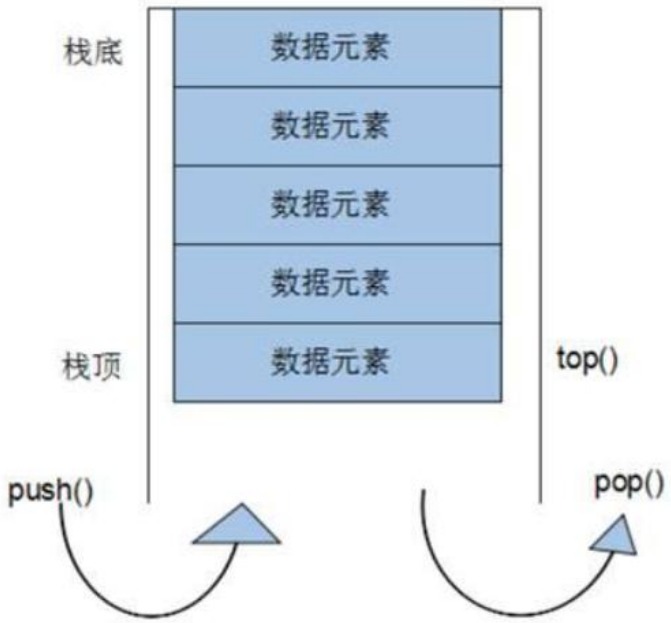
- `at(int idx);` //返回索引idx所指的数据
- `operator[];` //返回索引idx所指的数据
- `front();` //返回容器中第一个数据元素
- `back();` //返回容器中最后一个数据元素

- 利用算法实现对deque容器进行排序

算法：

- `sort(iterator beg, iterator end)` //对beg和end区间内元素进行排序

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



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

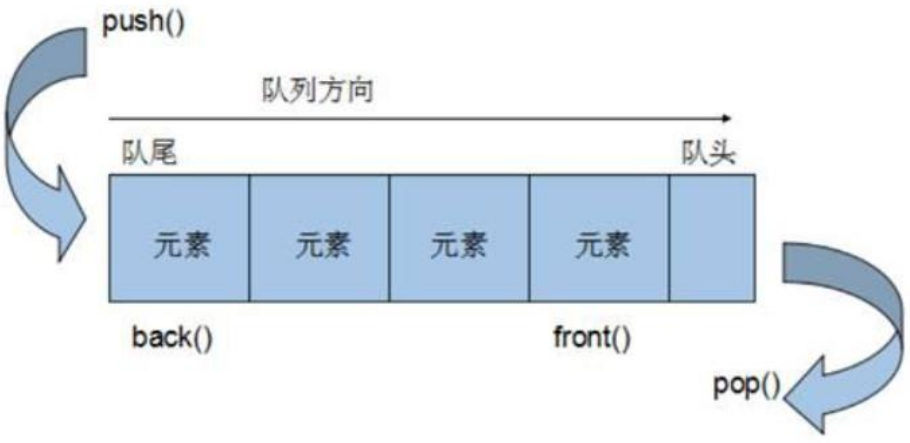
数据存取：

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

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



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：

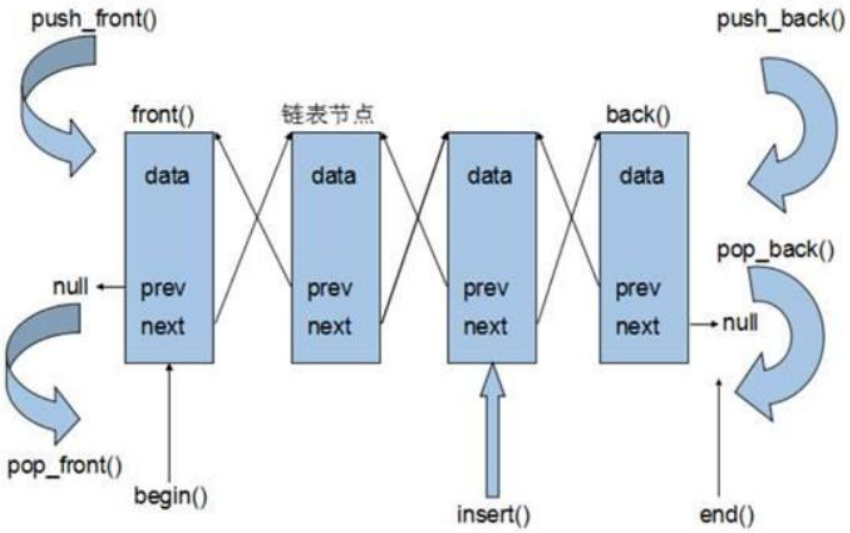
- `empty();` //判断堆栈是否为空
- `size();` //返回栈的大小

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



- 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

- 创建list容器

  函数原型：

  - `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。

- 给list容器进行赋值，以及交换list容器

  函数原型：

  - `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。

- 对list容器的大小进行操作

  函数原型：

  - `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。

- 对list容器进行数据的插入和删除

函数原型：

- 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。

- 对list容器中数据进行存取

函数原型：

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

- 将容器中的元素反转，以及将容器中的数据进行排序

函数原型：

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

3.8.1 set基本概念

简介：

- 所有元素都会在插入时自动被排序

本质：

- set/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

功能描述：创建set容器以及赋值

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

功能描述：

- 统计set容器大小以及交换set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

- set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(elem);` //删除容器中值为elem的元素。

功能描述：

- 对set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

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 );`

3.9.1 map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

函数原型：

构造：

- `map<T1, T2> mp;` //map默认构造函数:
- `map(const map &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。

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