

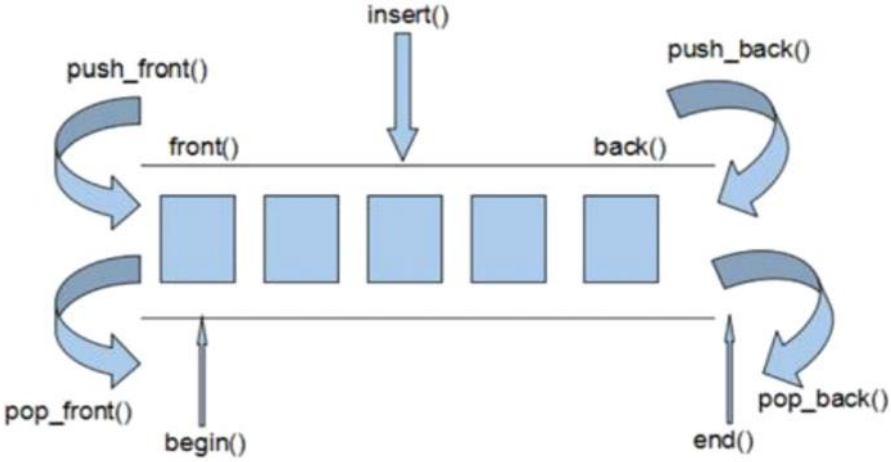
STL-常用容器

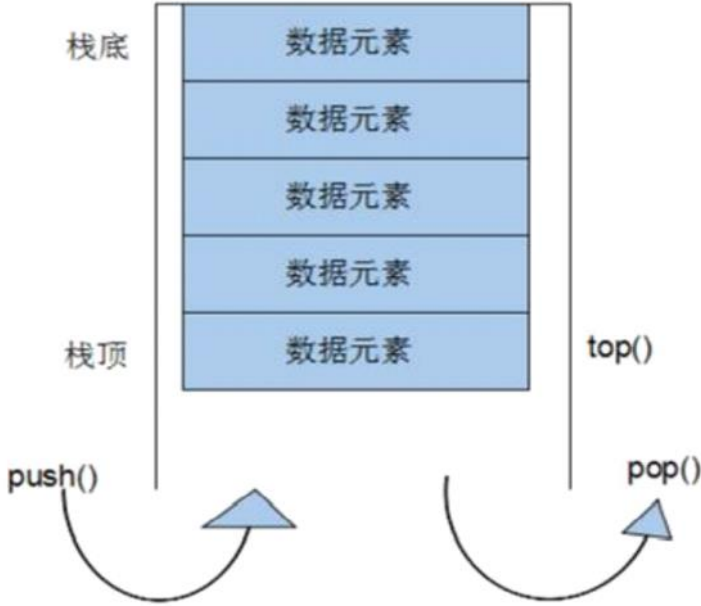
2022年1月9日 22:04

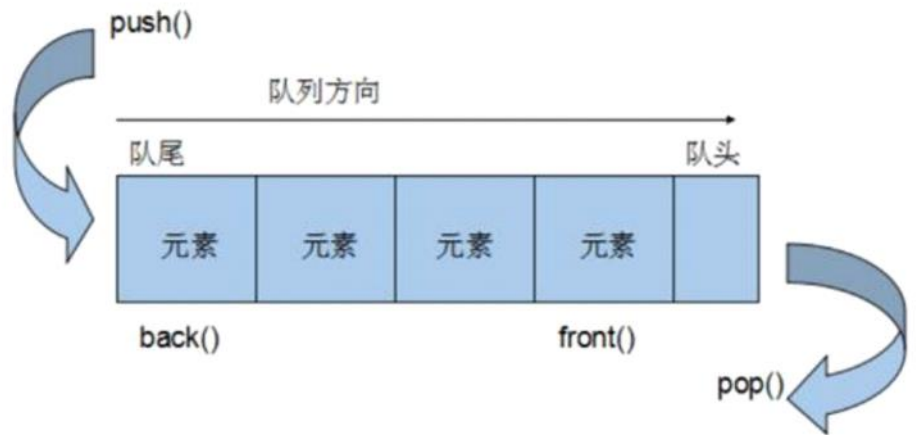
STL- 常用容器

string容器	基本概念	string是C++风格的字符串，而string本质上是一个类 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	<code>string();</code> //创建一个空的字符串 例如: <code>string str;</code> <code>string(const char* s);</code> //使用字符串s初始化 <code>string(const string& str);</code> //使用一个string对象初始化另一个string对象 <code>string(int n, char c);</code> //使用n个字符c初始化
	赋值操作	<code>string& operator=(const char* s);</code> //char*类型字符串 赋值给当前的字符串 <code>str1 = "hello world";</code> <code>string& operator=(const string &s);</code> //把字符串s赋给当前的字符串 <code>str2 = str1;</code> <code>string& operator=(char c);</code> //字符赋值给当前的字符串 <code>str3 = 'a';</code> <code>string& assign(const char *s);</code> //把字符串s赋给当前的字符串 <code>str4.assign("hello c++");</code> <code>string& assign(const char *s, int n);</code> //把字符串s的前n个字符赋给当前的字符串 <code>str5.assign("hello c++",5);</code> <code>string& assign(const string &s);</code> //把字符串s赋给当前字符串 <code>str6.assign(str5);</code> <code>string& assign(int n, char c);</code> //用n个字符c赋给当前字符串 <code>str7.assign(5, 'x');</code>
	字符串拼接	<code>string& operator+=(const char* str);</code> //重载+=操作符 <code>string& operator+=(const char c);</code> //重载+=操作符 <code>string& operator+=(const string& str);</code> //重载+=操作符 <code>string& append(const char *s);</code> //把字符串s连接到当前字符串结尾 <code>string& append(const char *s, int n);</code> //把字符串s的前n个字符连接到当前字符串结尾 <code>string& append(const string &s);</code> //同operator+=(const string& str) <code>string& append(const string &s, int pos, int n);</code> //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	<code>int find(const string& str, int pos = 0) const;</code> //查找str第一次出现位置,从pos开始查找 <code>int pos = str1.find("de");</code> //find查找是从左往右, rfind从右往左 <code>int find(const char* s, int pos = 0) const;</code> //查找s第一次出现位置,从pos开始查找 <code>int find(const char* s, int pos, int n) const;</code> //从pos位置查找s的前n个字符第一次位置 <code>int find(const char c, int pos = 0) const;</code> //查找字符c第一次出现位置 <code>int rfind(const string& str, int pos = npos) const;</code> //查找str最后一次位置,从pos开始查找 <code>int pos = str1.rfind("de");</code> <code>int rfind(const char* s, int pos = npos) const;</code> //查找s最后一次出现位置,从pos开始查找 <code>int rfind(const char* s, int pos, int n) const;</code> //从pos查找s的前n个字符最后一次位置 <code>int rfind(const char c, int pos = 0) const;</code> //查找字符c最后一次出现位置 <code>string& replace(int pos, int n, const string& str);</code> //替换从pos开始n个字符为字符串str <code>str1.replace(1, 3, "1111");</code> <code>string& replace(int pos, int n,const char* s);</code> //替换从pos开始的n个字符为字符串s find找到字符串后返回查找的第一个字符位置，找不到返回-1 replace在替换时，要指定从哪个位置起，多少个字符，替换成什么样的字符串
	字符串比较	<code>int compare(const string &s) const;</code> //与字符串s比较 <code>int ret = s1.compare(s2);</code> <code>int compare(const char *s) const;</code> //与字符串s比较 字符串对比主要是用于比较两个字符串是否相等，判断谁大谁小的意义并不是很大
	字符存取	<code>char& operator[](int n);</code> //通过[]方式取字符

		<pre>str[i] char& at(int n); //通过at方法获取字符 str.at(i)</pre>
插入和删除		<pre>string& insert(int pos, const char* s); //插入字符串 string& insert(int pos, const string& str); //插入字符串 str.insert(1, "111"); 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 str.erase(1, 3); //从1号位置开始3个字符 插入和删除的起始下标都是从0开始</pre>
string子串		<pre>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</pre>
vector容器	基本概念	<p>vector数据结构和数组非常相似，也称为单端数组</p> <p>vector与普通数组区别：</p> <p>不同之处在于数组是静态空间，而vector可以动态扩展</p> <p>动态扩展：并不是在原空间之后续接新空间，而是找更大的内存空间，然后将原数据拷贝到新空间，释放原空间</p> <p>vector容器的迭代器是支持随机访问的迭代器</p>
构造函数		<pre>vector 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。</pre>
赋值操作		<pre>vector& operator=(const vector &vec); //重载等号操作符 v2 = v1; assign(beg, end); //将[beg, end)区间中的数据拷贝赋值给本身。 v3.assign(v1.begin(), v1.end()); assign(n, elem); //将n个elem拷贝赋值给本身。 v4.assign(10, 100);</pre>
容量和大小		<pre>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0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</pre>
插入和删除		<pre>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 v1.insert(v1.begin(), 2, 1000); 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</pre>

	数据存取	<code>at(int idx);</code> //返回索引idx所指的数据 <code>v1.at(i)</code> <code>operator[];</code> //返回索引idx所指的数据 <code>v1[i]</code> <code>front();</code> //返回容器中第一个数据元素 <code>back();</code> //返回容器中最后一个数据元素
	互换容器	<code>swap(vec);</code> // 将vec与本身的元素互换 <code>swap</code> 可以使两个容器互换, 可以达到实用的收缩内存效果
	预留空间	<code>reserve(int len);</code> //容器预留len个元素长度, 预留位置不初始化, 元素不可访问。 如果数据量较大, 可以一开始利用 <code>reserve</code> 预留空间
deque容器	基本概念	<p>功能: 双端数组, 可以对头端进行插入删除操作</p> <p>deque与vector区别:</p> <p>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</p> <p>deque内部工作原理:</p> <p>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</p> <p>deque容器的迭代器也是支持随机访问的</p> 
	构造函数	<code>deque<T> deqT;</code> //默认构造形式 <code>deque<int> d1;</code> <code>deque(beg, end);</code> //构造函数将[beg, end)区间中的元素拷贝给本身。 <code>deque<int> d2(d1.begin(), d1.end());</code> <code>deque(n, elem);</code> //构造函数将n个elem拷贝给本身。 <code>deque<int> d3(10, 100);</code> <code>deque(const deque &deq);</code> //拷贝构造函数 <code>deque<int> d4 = d3;</code> deque容器和vector容器的构造方式几乎一致
	赋值操作	<code>deque& operator=(const deque &deq);</code> //重载等号操作符 <code>d2 = d1;</code> <code>assign(beg, end);</code> //将[beg, end)区间中的数据拷贝赋值给本身。 <code>assign(n, elem);</code> //将n个elem拷贝赋值给本身。
	容量和大小	<code>deque.empty();</code> //判断容器是否为空, 空返回1 <code>deque.size();</code> //返回容器中元素的个数 <code>deque.resize(num);</code> //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。 <code>deque.resize(num, elem);</code> //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。 <code>d1.resize(15, 1);</code> //如果容器变短, 则末尾超出容器长度的元素被删除。 deque没有容量的概念
	插入和删除	<code>push_back(elem);</code> //在容器尾部添加一个数据

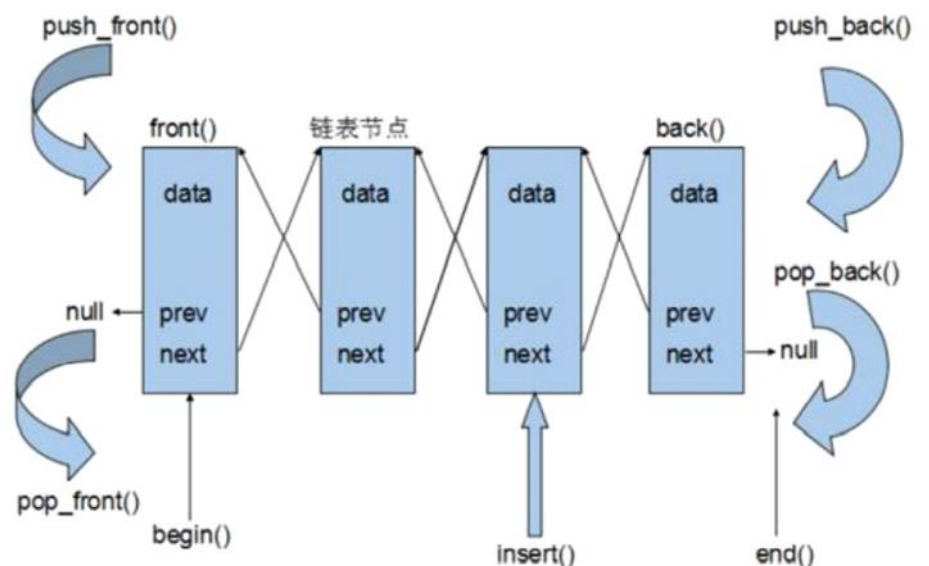
		<code>push_front(elem);</code> //在容器头部插入一个数据 <code>pop_back();</code> //删除容器最后一个数据 <code>pop_front();</code> //删除容器第一个数据 <code>insert(pos,elem);</code> //在pos位置插入一个elem元素的拷贝，返回新数据的位置。 <code>insert(pos,n,elem);</code> //在pos位置插入n个elem数据，无返回值。 <code>insert(pos,beg,end);</code> //在pos位置插入[beg,end)区间的数据，无返回值。 <code>clear();</code> //清空容器的所有数据 <code>erase(beg,end);</code> //删除[beg,end)区间的数据，返回下一个数据的位置。 <code>erase(pos);</code> //删除pos位置的数据，返回下一个数据的位置。
	数据存取	<code>at(int idx);</code> //返回索引idx所指的数据 <code>d.at(i)</code> <code>operator[];</code> //返回索引idx所指的数据 <code>front();</code> //返回容器中第一个数据元素 <code>d.front()</code> <code>back();</code> //返回容器中最后一个数据元素
stack容器	基本概念	<p>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 栈中进入数据称为 --- 入栈 push 栈中弹出数据称为 --- 出栈 pop</p> 
	构造函数	<code>stack<T> stk;</code> //stack采用模板类实现， stack对象的默认构造形式 <code>stack<int> s;</code> <code>stack(const stack &stk);</code> //拷贝构造函数
	赋值操作	<code>stack& operator=(const stack &stk);</code> //重载等号操作符
	数据存取	<code>push(elem);</code> //向栈顶添加元素 <code>s.push(10);</code> <code>pop();</code> //从栈顶移除第一个元素 <code>top();</code> //返回栈顶元素
	大小操作	<code>empty();</code> //判断堆栈是否为空 <code>size();</code> //返回栈的大小
queue 容器	基本概念	<p>Queue是一种先进先出(First In First Out,FIFO)的数据结构，它有两个出口 队列容器允许从一端新增元素，从另一端移除元素 队列中只有队头和队尾才可以被外界使用，因此队列不允许有遍历行为 队列中进数据称为 --- 入队 push 队列中出数据称为 --- 出队 pop</p>



构造函数	<code>queue<T> que;</code> //queue采用模板类实现, queue对象的默认构造形式 <code>queue(const queue &que);</code> //拷贝构造函数
赋值操作	<code>queue& operator=(const queue &que);</code> //重载等号操作符
数据存取	<code>push(elem);</code> //往队尾添加元素 <code>pop();</code> //从队头移除第一个元素 <code>back();</code> //返回最后一个元素 <code>front();</code> //返回第一个元素
大小操作	<code>empty();</code> //判断堆栈是否为空 <code>size();</code> //返回栈的大小

list容器

基本概念	<p>功能：将数据进行链式存储</p> <p>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</p> <p>链表的组成：链表由一系列结点组成</p> <p>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</p> <p>STL中的链表是一个双向循环链表</p> <p>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</p> <p>list的优点：</p> <p>采用动态存储分配，不会造成内存浪费和溢出</p> <p>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</p> <p>list的缺点：</p> <p>链表灵活，但是空间(指针域) 和 时间（遍历）额外耗费较大</p> <p>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。</p>
------	--



构造函数	<code>list<T> lst;</code> //list采用模板类实现,对象的默认构造形式: <code>list(beg,end);</code> //构造函数将[beg, end)区间中的元素拷贝给本身。
------	---

		<code>list(n,elem);</code> //构造函数将n个elem拷贝给本身。 <code>list(const list &lst);</code> //拷贝构造函数。 list构造方式同其他几个STL常用容器
赋值操作		<code>assign(beg, end);</code> //将[beg, end)区间中的数据拷贝赋值给本身。 <code>L3.assign(L2.begin(), L2.end());</code> <code>assign(n, elem);</code> //将n个elem拷贝赋值给本身。 <code>L4.assign(10, 100);</code> <code>list& operator=(const list &lst);</code> //重载等号操作符 <code>swap(lst);</code> //将lst与本身的元素互换。 <code>L1.swap(L2);</code>
大小操作		<code>size();</code> //返回容器中元素的个数 <code>empty();</code> //判断容器是否为空 <code>resize(num);</code> //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。 <code>resize(num, elem);</code> //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。
插入和删除		<code>push_back(elem);</code> //在容器尾部加入一个元素 <code>L.push_back(10);</code> <code>pop_back();</code> //删除容器中最后一个元素 <code>L.pop_back();</code> <code>push_front(elem);</code> //在容器开头插入一个元素 <code>pop_front();</code> //从容器开头移除第一个元素 <code>insert(pos,elem);</code> //在pos位置插elem元素的拷贝，返回新数据的位置。 <code>list<int>::iterator it = L.begin();</code> <code>L.insert(++it, 1000);</code> <code>insert(pos,n,elem);</code> //在pos位置插入n个elem数据，无返回值。 <code>insert(pos,beg,end);</code> //在pos位置插入[beg,end)区间的数据，无返回值。 <code>clear();</code> //移除容器的所有数据 <code>erase(beg,end);</code> //删除[beg,end)区间的数据，返回下一个数据的位置。 <code>erase(pos);</code> //删除pos位置的数据，返回下一个数据的位置。 <code>remove(elem);</code> //删除容器中所有与elem值匹配的元素。 <code>L.remove(10);</code>
数据存取		<code>front();</code> //返回第一个元素。 <code>back();</code> //返回最后一个元素。 list容器中不可以通过[]或者at方式访问数据
反转和排序		<code>reverse();</code> //反转链表 <code>sort();</code> //链表排序 <code>L.sort();</code> //默认的排序规则 从小到大 不支持sort(L)

set/ multiset 容器

基本概念	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。 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
构造函数	<code>set<T> st;</code> //默认构造函数： <code>set<int> s1;</code> <code>set(const set &st);</code> //拷贝构造函数
赋值操作	<code>set& operator=(const set &st);</code> //重载等号操作符 <code>s1.insert(10);</code> set容器插入数据时用insert，没有set.push(elm) set容器插入数据的数据会自动排序
大小操作	<code>size();</code> //返回容器中元素的数目 <code>empty();</code> //判断容器是否为空 <code>swap(st);</code> //交换两个集合容器 <code>s1.swap(s2);</code>

		<p>插入和删除</p> <pre> 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。 s1.erase(s1.begin()); 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。 </pre> <p>统计和查找</p> <pre> 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(); set<int>::iterator pos = s1.find(30); count(key); //统计key的元素个数 int num = s1.count(30); </pre> <p>pair对组创建</p> <pre> pair<type, type> p (value1, value2); pair<string, int> p(string("Tom"), 20); pair<type, type> p = make_pair(value1, value2); pair<string, int> p2 = make_pair("Jerry", 10); </pre>
map/ multimap容器	<p>基本概念</p> <p>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</p> <p>构造函数</p> <pre> map<T1, T2> mp; //map默认构造函数: map<int,int> m; map(const map &mp); //拷贝构造函数 map<int, int> m2(m); </pre> <p>赋值操作</p> <pre> map& operator=(const map &mp); //重载等号操作符 m.insert(pair<int, int>(1, 10)); </pre> <p>大小和交换</p> <pre> 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 m.swap(m2); </pre> <p>插入和删除</p> <pre> insert(elem); //在容器中插入元素。 //第一种插入方式 m.insert(pair<int, int>(1, 10)); //第二种插入方式 m.insert(make_pair(2, 20)); //第三种插入方式 m.insert(map<int, int>::value_type(3, 30)); //第四种插入方式 m[4] = 40; 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。 </pre> <p>查找和统计</p> <pre> 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 </pre>	