

07_19_第九章 _vector_string_适配器 器.md

小書匠

目录

第九章 vector string 适配器	1
vector 对象是如何增长的	1
string 的操作	1
string 的构造	1
改变 string 的其他方法	1
string 的搜索操作	2
compare 函数	3
数值转换	3
适配器机制	3

第九章 vector string 适配器

vector 对象是如何增长的

容器大小管理操作	概述
<code>c.shrink_to_fit</code>	将 <code>capacity()</code> 减少为 <code>size()</code> 相同大小(不保证一定退回内存空间)
<code>c.capacity</code>	不重新分配内存空间的话, c可以保存多少元素
<code>c.reserve</code>	分配至少能容纳 n 个元素的内存空间

- `reserve` 只有需要的内存空间超过当前容量是, 才会改变 `vector` 的容量, 它至少会分配与需求一样大的内存空间, 可能会更大
- `capacity` 和 `size` 的区别
 - `capacity` 是可以容纳的元素大小
 - `size` 是当前保存的元素的数量

string 的操作

string 的构造

构造string的其他方法	概述
<code>string s(cp,n)</code>	s是 cp指向的数组中的前n个字符的 拷贝 , 保证 cp 至少 n 个字符
<code>string s(s2,pos2)</code>	s是 string s2 从下标 pos2 开始的字符的拷贝。保证 <code>pos2 < s2.size()</code>
<code>string s(s2,pos2,len2)</code>	同上, len2 的值无论多大, 最多拷贝到 s2 结束(这里 len2 可以很大)

substr 操作	概述
<code>s.substr(pos,n)</code>	返回一个string, 包含 s 中从 pos 开始的 n 个字符的拷贝, 他们的默认值, 返回的恰好是 s 本身的拷贝

改变 string 的其他方法

修改 string 的操作	概述
<code>s.insert(pos,args)</code>	在pos之前插入args指定的字符。pos可以是一个下标或迭代器。下标版本返回指向 s 的引用, 迭代器版本返回指向第一个插入字符的迭代器。
<code>s.erase(pos,len)</code>	删除从位置pos开始的 len 个字符。如果 len 被省略, 则删除从 pos 开始直至 s 末尾的所有字符。返回值是 s 的引用
<code>s.assign(args)</code>	全部替换成 args 指定的字符, 返回引用
<code>s.append(pos,args)</code>	直接在后面追加, 返回引用
<code>s.replace(range,args)</code>	删除 s 中 range 内的字符, 替换成 args 指定的字符, range 是一个下标加一个长度, 或者是一对指向 s 的迭代器。返回引用

args的形式	概述
<code>str</code>	字符串str
<code>str,pos,len</code>	str 中从 pos 开始最多 len 个字
<code>cp,len</code>	从cp指向的字符数组的前(最多)len 个字符
<code>cp</code>	cp 指向的, 以 \0 结尾的字符数组
<code>n,c</code>	n 个字符 c
<code>b,e</code>	迭代器表示的范围
<code>\{...\}</code>	花括号包围, 用 逗号 分隔的, 字符列表

- args可以是以上之一的形势, append 和 assign 可以用所有形式
- 对于 replace 和 insert 这两个函数, args 不一定是通用的, 可以直接查书上的表

```

162     std::string::size_type;
163
164     std::string str = "123456";
165     std::string &str2 = str;
166     str.insert(str.begin(), str2);
167 }
168

```

没有与参数列表匹配的 重载函数 "std::basic_string<_Elem, _Traits, _Alloc>::insert [其中 _Elem=char, _Traits=, 参数类型为: (std::_String_iterator<std::_String_val<std::conditional_t<true, std::_Simple_types<cha 对象类型是: std::string

1

- 这个就是没有对应的接口

string 的搜索操作

- 搜索返回的值类型是 `string::size_type`, 这个值其实是一个 unsigned 类型, 所以一般不用 int 这种带符号类型来保存
- 如果找不到, 会返回 `string::npos`, 这个其实是对一个 unsigned 类型值, 初始化为 -1 的值, 可以理解成最大的大小

string 搜索操作	概述
<code>s.find(args)</code>	查找s中 args 第一次出现的位置
<code>s.rfind(args)</code>	查找s中 args 最后一次出现的位置
<code>s.find_first_of(args)</code>	查找s中 args 中任何一个字符 第一次出现的位置
<code>s.find_last_of(args)</code>	查找s中 args 中任何一个字符 最后一次出现的位置
<code>s.find_first_not_of(args)</code>	查找s中 第一个不在 args 的字符出现的位置
<code>s.find_last_not_of(args)</code>	查找s中 最后一个不在 args 的字符出现的位置

args的形式	概述
<code>c,pos</code>	从s中位置 pos 开始查找字符 c, pos默认 0
<code>s2,pos</code>	从s中位置 pos 开始查找字符串 s2, pos默认 0
<code>cp,pos</code>	从s中位置 pos 开始查找 cp 指向的, 以\0 结尾的C风格字符串, pos默认 0
<code>cp,n,pos</code>	从s中位置 pos 开始查找 cp 指向的, 前n个字符, pos默认 0

- 写循环搜索, 记得要**递增pos**, 因为搜索从pos开始, 如果找到了, 就会返回 pos

compare 函数

- 这个函数就是一个字符串比较

数值转换

- 主要涉及的是, string 向 int, float 等的数值转换(支持正负号开头)

string 和 数值之间的转换	概述
<code>to_string(val)</code>	返回 val 的 string 表示
<code>stoi(s,p,b)</code>	p 是下标, 默认是 0, b 是基数, 默认 10。
<code>stol(s,p,b)</code>	
<code>stoul(s,p,b)</code>	
<code>stoll(s,p,b)</code>	
<code>stoull(s,p,b)</code>	
<code>stof(s,p)</code>	返回浮点数系
<code>stod(s,p)</code>	
<code>stold(s,p)</code>	

适配器机制

适配器(adaptor) 是一个标准库的通用概念, 他是一种机制, 使得某种底层容器, 表现得跟适配器一样。适配器有, **栈(stack)**, **队列(queue)**, **优先队列(priority_queue)**

他们的默认实现分是

- stack: deque
- queue: deque
- priority_queue :vector

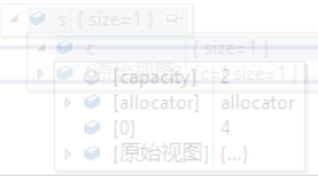
```
std::string &str2 = str; Ty, typename _Container = deque < _Ty >>  
CLASS TEMPLATE stack  
stack<int,
```

2

这里的就是默认 deque

我们也可以用其他的底层的数据结构来实现相应的适配器：

```
stack<int, vector<int>>> s;  
s.push(4);  
s.push(2);  
s.pop();
```



3

核心:适配器机制 = 底层数据结构, 通过接口的上层封装