1. 标准C++中的string类的用法总结
2. 相信使用过MFC编程的朋友对CString这个类的印象应该非常深刻吧？的确，MFC中的CString类使用起来真的非常的方便好用。但是如果离开了MFC框架，还有没有这样使用起来非常方便的类呢？答案是肯定的。也许有人会说，即使不用MFC框架，也可以想办法使用MFC中的API，具体的操作方法在本文最后给出操作方法。其实，可能很多人很可能会忽略掉标准C++中string类的使用。标准C++中提供的string类得功能也是非常强大的，一般都能满足我们开发项目时使用。现将具体用法的一部分罗列如下，只起一个抛砖引玉的作用吧，好了，废话少说，直接进入正题吧！
3. 要想使用标准C++中string类，必须要包含
4. #include <string>// 注意是<string>，不是<string.h>，带.h的是C语言中的头文件
5. **using**  std::string;
6. **using**  std::wstring;
7. 或
8. **using** **namespace** std;
9. 下面你就可以使用string/wstring了，它们两分别对应着**char**和**wchar\_t**。
11. string和wstring的用法是一样的，以下只用string作介绍：

14. string类的构造函数：
16. string(**const** **char** \*s);    //用c字符串s初始化
17. string(**int** n,**char** c);     //用n个字符c初始化
18. 此外，string类还支持默认构造函数和复制构造函数，如string s1；string s2="hello"；都是正确的写法。当构造的string太长而无法表达时会抛出length\_error异常 ；

21. string类的字符操作：
22. **const** **char** &operator[](**int** n)**const**;
23. **const** **char** &at(**int** n)**const**;
24. **char** &operator[](**int** n);
25. **char** &at(**int** n);
26. operator[]和at()均返回当前字符串中第n个字符的位置，但at函数提供范围检查，当越界时会抛出out\_of\_range异常，下标运算符[]不提供检查访问。
27. **const** **char** \*data()**const**;//返回一个非null终止的c字符数组
28. **const** **char** \*c\_str()**const**;//返回一个以null终止的c字符串
29. **int** copy(**char** \*s, **int** n, **int** pos = 0) **const**;//把当前串中以pos开始的n个字符拷贝到以s为起始位置的字符数组中，返回实际拷贝的数目

32. string的特性描述:
33. **int** capacity()**const**;    //返回当前容量（即string中不必增加内存即可存放的元素个数）
34. **int** max\_size()**const**;    //返回string对象中可存放的最大字符串的长度
35. **int** size()**const**;        //返回当前字符串的大小
36. **int** length()**const**;       //返回当前字符串的长度
37. **bool** empty()**const**;        //当前字符串是否为空
38. **void** resize(**int** len,**char** c);//把字符串当前大小置为len，并用字符c填充不足的部分
40. string类的输入输出操作:
41. string类重载运算符operator>>用于输入，同样重载运算符operator<<用于输出操作。
42. 函数getline(istream &in,string &s);用于从输入流in中读取字符串到s中，以换行符'\n'分开。
44. string的赋值：
45. string &operator=(**const** string &s);//把字符串s赋给当前字符串
46. string &assign(**const** **char** \*s);//用c类型字符串s赋值
47. string &assign(**const** **char** \*s,**int** n);//用c字符串s开始的n个字符赋值
48. string &assign(**const** string &s);//把字符串s赋给当前字符串
49. string &assign(**int** n,**char** c);//用n个字符c赋值给当前字符串
50. string &assign(**const** string &s,**int** start,**int** n);//把字符串s中从start开始的n个字符赋给当前字符串
51. string &assign(const\_iterator first,const\_itertor last);//把first和last迭代器之间的部分赋给字符串
53. string的连接：
54. string &operator+=(**const** string &s);//把字符串s连接到当前字符串的结尾
55. string &append(**const** **char** \*s);            //把c类型字符串s连接到当前字符串结尾
56. string &append(**const** **char** \*s,**int** n);//把c类型字符串s的前n个字符连接到当前字符串结尾
57. string &append(**const** string &s);    //同operator+=()
58. string &append(**const** string &s,**int** pos,**int** n);//把字符串s中从pos开始的n个字符连接到当前字符串的结尾
59. string &append(**int** n,**char** c);        //在当前字符串结尾添加n个字符c
60. string &append(const\_iterator first,const\_iterator last);//把迭代器first和last之间的部分连接到当前字符串的结尾

63. string的比较：
64. **bool** operator==(**const** string &s1,**const** string &s2)**const**;//比较两个字符串是否相等
65. 运算符">","<",">=","<=","!="均被重载用于字符串的比较；
66. **int** compare(**const** string &s) **const**;//比较当前字符串和s的大小
67. **int** compare(**int** pos, **int** n,**const** string &s)**const**;//比较当前字符串从pos开始的n个字符组成的字符串与s的大小
68. **int** compare(**int** pos, **int** n,**const** string &s,**int** pos2,**int** n2)**const**;//比较当前字符串从pos开始的n个字符组成的字符串与s中
70. //pos2开始的n2个字符组成的字符串的大小
71. **int** compare(**const** **char** \*s) **const**;
72. **int** compare(**int** pos, **int** n,**const** **char** \*s) **const**;
73. **int** compare(**int** pos, **int** n,**const** **char** \*s, **int** pos2) **const**;
74. compare函数在>时返回1，<时返回-1，==时返回0

77. string的子串：
78. string substr(**int** pos = 0,**int** n = npos) **const**;//返回pos开始的n个字符组成的字符串
80. string的交换：
81. **void** swap(string &s2);    //交换当前字符串与s2的值

84. string类的查找函数：
85. **int** find(**char** c, **int** pos = 0) **const**;//从pos开始查找字符c在当前字符串的位置
86. **int** find(**const** **char** \*s, **int** pos = 0) **const**;//从pos开始查找字符串s在当前串中的位置
87. **int** find(**const** **char** \*s, **int** pos, **int** n) **const**;//从pos开始查找字符串s中前n个字符在当前串中的位置
88. **int** find(**const** string &s, **int** pos = 0) **const**;//从pos开始查找字符串s在当前串中的位置
89. //查找成功时返回所在位置，失败返回string::npos的值
90. **int** rfind(**char** c, **int** pos = npos) **const**;//从pos开始从后向前查找字符c在当前串中的位置
91. **int** rfind(**const** **char** \*s, **int** pos = npos) **const**;
92. **int** rfind(**const** **char** \*s, **int** pos, **int** n = npos) **const**;
93. **int** rfind(**const** string &s,**int** pos = npos) **const**;
94. //从pos开始从后向前查找字符串s中前n个字符组成的字符串在当前串中的位置，成功返回所在位置，失败时返回string::npos的值
95. **int** find\_first\_of(**char** c, **int** pos = 0) **const**;//从pos开始查找字符c第一次出现的位置
96. **int** find\_first\_of(**const** **char** \*s, **int** pos = 0) **const**;
97. **int** find\_first\_of(**const** **char** \*s, **int** pos, **int** n) **const**;
98. **int** find\_first\_of(**const** string &s,**int** pos = 0) **const**;
99. //从pos开始查找当前串中第一个在s的前n个字符组成的数组里的字符的位置。查找失败返回string::npos
100. **int** find\_first\_not\_of(**char** c, **int** pos = 0) **const**;
101. **int** find\_first\_not\_of(**const** **char** \*s, **int** pos = 0) **const**;
102. **int** find\_first\_not\_of(**const** **char** \*s, **int** pos,**int** n) **const**;
103. **int** find\_first\_not\_of(**const** string &s,**int** pos = 0) **const**;
104. //从当前串中查找第一个不在串s中的字符出现的位置，失败返回string::npos
105. **int** find\_last\_of(**char** c, **int** pos = npos) **const**;
106. **int** find\_last\_of(**const** **char** \*s, **int** pos = npos) **const**;
107. **int** find\_last\_of(**const** **char** \*s, **int** pos, **int** n = npos) **const**;
108. **int** find\_last\_of(**const** string &s,**int** pos = npos) **const**;
109. **int** find\_last\_not\_of(**char** c, **int** pos = npos) **const**;
110. **int** find\_last\_not\_of(**const** **char** \*s, **int** pos = npos) **const**;
111. **int** find\_last\_not\_of(**const** **char** \*s, **int** pos, **int** n) **const**;
112. **int** find\_last\_not\_of(**const** string &s,**int** pos = npos) **const**;
113. //find\_last\_of和find\_last\_not\_of与find\_first\_of和find\_first\_not\_of相似，只不过是从后向前查找

116. string类的替换函数：
117. string &replace(**int** p0, **int** n0,**const** **char** \*s);//删除从p0开始的n0个字符，然后在p0处插入串s
118. string &replace(**int** p0, **int** n0,**const** **char** \*s, **int** n);//删除p0开始的n0个字符，然后在p0处插入字符串s的前n个字符
119. string &replace(**int** p0, **int** n0,**const** string &s);//删除从p0开始的n0个字符，然后在p0处插入串s
120. string &replace(**int** p0, **int** n0,**const** string &s, **int** pos, **int** n);//删除p0开始的n0个字符，然后在p0处插入串s中从pos开始的n个字符
121. string &replace(**int** p0, **int** n0,**int** n, **char** c);//删除p0开始的n0个字符，然后在p0处插入n个字符c
122. string &replace(iterator first0, iterator last0,**const** **char** \*s);//把[first0，last0）之间的部分替换为字符串s
123. string &replace(iterator first0, iterator last0,**const** **char** \*s, **int** n);//把[first0，last0）之间的部分替换为s的前n个字符
124. string &replace(iterator first0, iterator last0,**const** string &s);//把[first0，last0）之间的部分替换为串s
125. string &replace(iterator first0, iterator last0,**int** n, **char** c);//把[first0，last0）之间的部分替换为n个字符c
126. string &replace(iterator first0, iterator last0,const\_iterator first, const\_iterator last);//把[first0，last0）之间的部分替换成[first，last）之间的字符串

129. string类的插入函数：
130. string &insert(**int** p0, **const** **char** \*s);
131. string &insert(**int** p0, **const** **char** \*s, **int** n);
132. string &insert(**int** p0,**const** string &s);
133. string &insert(**int** p0,**const** string &s, **int** pos, **int** n);
134. //前4个函数在p0位置插入字符串s中pos开始的前n个字符
135. string &insert(**int** p0, **int** n, **char** c);//此函数在p0处插入n个字符c
136. iterator insert(iterator it, **char** c);//在it处插入字符c，返回插入后迭代器的位置
137. **void** insert(iterator it, const\_iterator first, const\_iterator last);//在it处插入[first，last）之间的字符
138. **void** insert(iterator it, **int** n, **char** c);//在it处插入n个字符c

141. string类的删除函数
142. iterator erase(iterator first, iterator last);//删除[first，last）之间的所有字符，返回删除后迭代器的位置
143. iterator erase(iterator it);//删除it指向的字符，返回删除后迭代器的位置
144. string &erase(**int** pos = 0, **int** n = npos);//删除pos开始的n个字符，返回修改后的字符串

147. string类的迭代器处理：
148. string类提供了向前和向后遍历的迭代器iterator，迭代器提供了访问各个字符的语法，类似于指针操作，迭代器不检查范围。
149. 用string::iterator或string::const\_iterator声明迭代器变量，const\_iterator不允许改变迭代的内容。常用迭代器函数有：
150. const\_iterator begin()**const**;
151. iterator begin();                //返回string的起始位置
152. const\_iterator end()**const**;
153. iterator end();                    //返回string的最后一个字符后面的位置
154. const\_iterator rbegin()**const**;
155. iterator rbegin();                //返回string的最后一个字符的位置
156. const\_iterator rend()**const**;
157. iterator rend();                    //返回string第一个字符位置的前面
158. rbegin和rend用于从后向前的迭代访问，通过设置迭代器string::reverse\_iterator,string::const\_reverse\_iterator实现