昵称：aminxu
园龄：5年2个月
粉丝：10
关注：7
+加关注

< 2019年3月 >

| 日 | 一 | 二 | 三 | 四 | 五 | 六 |
|---|---|---|---|---|---|---|
| 24 | 25 | 26 | 27 | 28 | 1 | 2 |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 1 | 2 | 3 | 4 | 5 | 6 |

搜索

找找看

谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签
更多链接

随笔分类

c++(10)
matlab(19)
ogre(5)
opencv(1)
opengl(15)
vs(13)
web开发(2)
辐射标定(1)
计算机(1)
图形学(7)
相机标定(3)

随笔档案

2016年10月 (3)
2016年8月 (4)
2016年1月 (1)
2015年8月 (15)
2015年7月 (28)
2015年6月 (7)
2015年5月 (9)
2015年4月 (4)
2015年3月 (1)
2015年2月 (8)
2015年1月 (5)
2014年12月 (1)
2014年11月 (1)

最新评论

1. Re:crontab 详细用法 定时任务
@小渣渣。对的看的挺仔细，那里确实是个坑...
--云域之
2. Re:crontab 详细用法 定时任务
这是个坑 我被坑过
--小渣渣。
3. Re:crontab 详细用法 定时任务

## 标准C++中的string类的用法总结（转）

转自：http://www.cnblogs.com/xFreedom/archive/2011/05/16/2048037.html

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！

要想使用标准C++中string类，必须要包含

#include <string>// 注意是<string>，不是<string.h>，带.h的是C语言中的头文件

using std::string;

using std::wstring;

或

using namespace std;

下面你就可以使用string/wstring了，它们两分别对应着char和wchar_t。

string和wstring的用法是一样的，以下只用string作介绍：

# 构造函数(Constructors)

*语法:*

```
string();
string( size_type length, char ch );
string( const char *str );
string( const char *str, size_type length );
string( string &str, size_type index, size_type length );
string( input_iterator start, input_iterator end );
```

字符串的构造函数创建一个新字符串，包括:

- 以length为长度的ch的拷贝（即length个ch）
- 以str为初值 (长度任意),
- 以index为索引开始的子串，长度为length, 或者
- 以从start到end的元素为初值.

例如,

```
string str1( 5, 'c' );
string str2( "Now is the time..." );
string str3( str2, 11, 4 );
cout << str1 << endl;
cout << str2 << endl;
cout << str3 << endl;
```

显示

```
ccccc
Now is the time...
time
```

## 操作符(Operators)

*语法:*

```
==
>
<
>=
<=
!=
+
+=
[]
```

你可以用 ==, >, <, >=, <=, and !=比较字符串. 可以用 + 或者 += 操作符连接两个字符串, 并且可以用[]获取特定的字符.

*相关主题:*
**at(), compare().**

---

## 添加文本(append)

*语法:*

```
basic_string &append( const basic_string &str );
basic_string &append( const char *str );
basic_string &append( const basic_string &str, size_type index, size_type len );
basic_string &append( const char *str, size_type num );
basic_string &append( size_type num, char ch );
basic_string &append( input_iterator start, input_iterator end );
```

append() 函数可以完成以下工作:

- 在字符串的末尾添加str,
- 在字符串的末尾添加str的子串,子串以index索引开始，长度为len
- 在字符串的末尾添加str中的num个字符,
- 在字符串的末尾添加num个字符ch,
- 在字符串的末尾添加以迭代器start和end表示的字符序列.

例如以下代码:

```
string str = "Hello World";
str.append( 10, '!' );
cout << str << endl;
```

显示

```
Hello World!!!!!!!!!!
```

*相关主题:*
**+ 操作符**

---

## 赋值(assign)

*语法:*

```
basic_string &assign( const basic_string &str );
basic_string &assign( const char *str );
basic_string &assign( const char *str, size_type num );
basic_string &assign( const basic_string &str, size_type index, size_type len );
basic_string &assign( size_type num, char ch );
```

函数以下列方式赋值:

- 用str为字符串赋值,

- 用str的开始num个字符为字符串赋值,
- 用str的子串为字符串赋值,子串以index索引开始，长度为len
- 用num个字符ch为字符串赋值.

例如以下代码:

```
string str1, str2 = "War and Peace";
str1.assign( str2, 4, 3 );
cout << str1 << endl;
```

显示

```
and
```

---

## at

*语法:*

```
reference at( size_type index );
```

at()函数返回一个引用，指向在index位置的字符. 如果index不在字符串范围内, at() 将报告"out of range"错误，并抛出**out_of_range**异常。 比如下列代码:

```
string text = "ABCDEF";
char ch = text.at( 2 );
```

显示字符 'C'.

*相关主题:*
**[]操作符**

---

## begin

*语法:*

```
iterator begin();
```

begin()函数返回一个迭代器,指向字符串的第一个元素.

*相关主题:*
**end()**

---

## c_str

*语法:*

```
const char *c_str();
```

c_str()函数返回一个指向正规C字符串的指针, 内容与本字符串相同.

*相关主题:*
**[] 操作符**

---

## 容量(capacity)

*语法:*

```
size_type capacity();
```

capacity()函数返回在重新申请更多的空间前字符串可以容纳的字符数. 这个数字至少与 size()一样大.

*相关主题:*
**max_size()**, **reserve()**, **resize()**, **size()**,

---

## 比较(compare)

*语法:*

```
  int compare( const basic_string &str );
  int compare( const char *str );
  int compare( size_type index, size_type length, const basic_string &str );
  int compare( size_type index, size_type length, const basic_string &str, size_type
index2,
  size_type length2 );
  int compare( size_type index, size_type length, const char *str, size_type length2 );
```

compare()函数以多种方式比较本字符串和str,返回:

| 返回值 | 情况 |
| --- | --- |
| 小于零 | this < str |
| 零 | this == str |
| 大于零 | this > str |

不同的函数:

- 比较自己和str,
- 比较自己的子串和str,子串以index索引开始，长度为length
- 比较自己的子串和str的子串，其中index2和length2引用str，index和length引用自己
- 比较自己的子串和str的子串，其中str的子串以索引0开始，长度为length2，自己的子串以index开始，长度为length

*相关主题:*
**操作符**

---

## 拷贝(copy)

*语法:*

```
size_type copy( char *str, size_type num, size_type index );
```

copy()函数拷贝自己的num个字符到str中（从索引index开始）。返回值是拷贝的字符数

---

## data

*语法:*

```
const char *data();
```

data()函数返回指向自己的第一个字符的指针.

*相关主题:*
**c_str()**

---

## empty

*语法:*

```
bool empty();
```

如果字符串为空则empty()返回真(true)，否则返回假(false).

---

## end

*语法:*

```
iterator end();
```

end()函数返回一个迭代器，指向字符串的末尾(最后一个字符的下一个位置).

*相关主题:*
**begin()**

## 删除(erase)

*语法:*

```
iterator erase( iterator pos );
iterator erase( iterator start, iterator end );
basic_string &erase( size_type index = 0, size_type num = npos );
```

erase()函数可以:

- 删除pos指向的字符, 返回指向下一个字符的迭代器,
- 删除从start到end的所有字符, 返回一个迭代器,指向被删除的最后一个字符的下一个位置
- 删除从index索引开始的num个字符, 返回***this.**

参数*index* 和 *num* 有默认值, 这意味着erase()可以这样调用: 只带有*index*以删除index后的所有字符, 或者不带有任何参数以删除所有字符. 例如:

```
string s("So, you like donuts, eh? Well, have all the donuts in the world!");
cout << "The original string is '" << s << "'" << endl;

s.erase( 50, 14 );
cout << "Now the string is '" << s << "'" << endl;

s.erase( 24 );
cout << "Now the string is '" << s << "'" << endl;

s.erase();
cout << "Now the string is '" << s << "'" << endl;
```

将显示

```
The original string is 'So, you like donuts, eh? Well, have all the donuts in the world!'
Now the string is 'So, you like donuts, eh? Well, have all the donuts'
Now the string is 'So, you like donuts, eh?'
Now the string is ''
```

## 查找(find)

*语法:*

```
size_type find( const basic_string &str, size_type index );
size_type find( const char *str, size_type index );
size_type find( const char *str, size_type index, size_type length );
size_type find( char ch, size_type index );
```

find()函数:

- 返回str在字符串中第一次出现的位置（从index开始查找）。如果没找到则返回**string::npos**,
- 返回str在字符串中第一次出现的位置（从index开始查找，长度为length）。如果没找到就返回**string::npos**,
- 返回字符ch在字符串中第一次出现的位置（从index开始查找）。如果没找到就返回**string::npos**

例如,

```
string str1( "Alpha Beta Gamma Delta" );
unsigned int loc = str1.find( "Omega", 0 );
if( loc != string::npos )
  cout << "Found Omega at " << loc << endl;
else
  cout << "Didn't find Omega" << endl;
```

## find_first_of

*语法:*

```
size_type find_first_of( const basic_string &str, size_type index = 0 );
size_type find_first_of( const char *str, size_type index = 0 );
size_type find_first_of( const char *str, size_type index, size_type num );
size_type find_first_of( char ch, size_type index = 0 );
```

find_first_of()函数:

- 查找在字符串中第一个与str中的某个字符匹配的字符，返回它的位置。搜索从index开始，如果没找到就返回**string::npos**
- 查找在字符串中第一个与str中的某个字符匹配的字符，返回它的位置。搜索从index开始，最多搜索num个字符。如果没找到就返回**string::npos**,
- 查找在字符串中第一个与ch匹配的字符，返回它的位置。搜索从index开始。

*相关主题:*
**find()**

---

## find_first_not_of

*语法:*

```
size_type find_first_not_of( const basic_string &str, size_type index = 0 );
size_type find_first_not_of( const char *str, size_type index = 0 );
size_type find_first_not_of( const char *str, size_type index, size_type num );
size_type find_first_not_of( char ch, size_type index = 0 );
```

find_first_not_of()函数:

- 在字符串中查找第一个与str中的字符都不匹配的字符，返回它的位置。搜索从index开始。如果没找到就返回**string::nops**
- 在字符串中查找第一个与str中的字符都不匹配的字符，返回它的位置。搜索从index开始，最多查找num个字符。如果没找到就返回**string::nops**
- 在字符串中查找第一个与ch不匹配的字符，返回它的位置。搜索从index开始。如果没找到就返回**string::nops**

*相关主题:*
**find()**

---

## find_last_of

*语法:*

```
size_type find_last_of( const basic_string &str, size_type index = npos );
size_type find_last_of( const char *str, size_type index = npos );
size_type find_last_of( const char *str, size_type index, size_type num );
size_type find_last_of( char ch, size_type index = npos );
```

find_last_of()函数:

- 在字符串中查找最后一个与str中的某个字符匹配的字符，返回它的位置。搜索从index开始。如果没找到就返回**string::nops**
- 在字符串中查找最后一个与str中的某个字符匹配的字符，返回它的位置。搜索从index开始，最多搜索num个字符。如果没找到就返回**string::nops**
- 在字符串中查找最后一个与ch匹配的字符，返回它的位置。搜索从index开始。如果没找到就返回**string::nops**

*相关主题:*
**find()**

---

## find_last_not_of

*语法:*

```
size_type find_last_not_of( const basic_string &str, size_type index = npos );
size_type find_last_not_of( const char *str, size_type index = npos);
size_type find_last_not_of( const char *str, size_type index, size_type num );
size_type find_last_not_of( char ch, size_type index = npos );
```

find_last_not_of()函数:

- 在字符串中查找最后一个与str中的字符都不匹配的字符，返回它的位置。搜索从index开始。如果没找到就返回**string::nops**
- 在字符串中查找最后一个与str中的字符都不匹配的字符，返回它的位置。搜索从index开始，最多查找num个字符如果没找到就返回**string::nops**
- 在字符串中查找最后一个与ch不匹配的字符，返回它的位置。搜索从index开始。如果没找到就返回**string::nops**

*相关主题:*
**find()**

---

## get_allocator

*语法:*

```
allocator_type get_allocator();
```

get_allocator()函数返回本字符串的配置器.

---

## 插入(insert)

*语法:*

```
iterator insert( iterator i, const char &ch );
basic_string &insert( size_type index, const basic_string &str );
basic_string &insert( size_type index, const char *str );
basic_string &insert( size_type index1, const basic_string &str, size_type index2,
size_type num );
basic_string &insert( size_type index, const char *str, size_type num );
basic_string &insert( size_type index, size_type num, char ch );
void insert( iterator i, size_type num, const char &ch );
void insert( iterator i, iterator start, iterator end );
```

insert()函数的功能非常多:

- 在迭代器i表示的位置前面插入一个字符ch,
- 在字符串的位置index插入字符串str,
- 在字符串的位置index插入字符串str的子串(从*index2*开始，长num个字符),
- 在字符串的位置index插入字符串str的num个字符,
- 在字符串的位置index插入num个字符ch的拷贝,
- 在迭代器i表示的位置前面插入num个字符ch的拷贝,
- 在迭代器i表示的位置前面插入一段字符，从start开始，以end结束.

*相关主题:*
**replace()**

---

## 长度(length)

*语法:*

```
size_type length();
```

length()函数返回字符串的长度. 这个数字应该和size()返回的数字相同.

*相关主题:*
**size()**

---

## max_size

*语法:*

```
size_type max_size();
```

max_size()函数返回字符串能保存的最大字符数。

---

## rbegin

*语法:*

```
const reverse_iterator rbegin();
```

rbegin()返回一个逆向迭代器，指向字符串的最后一个字符。

*相关主题:*
**rend()**

---

## rend

*语法:*

```
const reverse_iterator rend();
```

rend()函数返回一个逆向迭代器，指向字符串的开头（第一个字符的前一个位置）。

*相关主题:*
**rbegin()**

---

## 替换(replace)

*语法:*

```
basic_string &replace( size_type index, size_type num, const basic_string &str );
basic_string &replace( size_type index1, size_type num1, const basic_string &str,
size_type index2,
size_type num2 );
basic_string &replace( size_type index, size_type num, const char *str );
basic_string &replace( size_type index, size_type num1, const char *str, size_type num2
);
basic_string &replace( size_type index, size_type num1, size_type num2, char ch );
basic_string &replace( iterator start, iterator end, const basic_string &str );
basic_string &replace( iterator start, iterator end, const char *str );
basic_string &replace( iterator start, iterator end, const char *str, size_type num );
basic_string &replace( iterator start, iterator end, size_type num, char ch );
```

replace()函数:

- 用str中的num个字符替换本字符串中的字符,从index开始
- 用str中的num2个字符（从index2开始）替换本字符串中的字符，从index1开始，最多num1个字符
- 用str中的num个字符（从index开始）替换本字符串中的字符
- 用str中的num2个字符（从index2开始）替换本字符串中的字符，从index1开始，num1个字符
- 用num2个ch字符替换本字符串中的字符，从index开始
- 用str中的字符替换本字符串中的字符,迭代器start和end指示范围
- 用str中的num个字符替换本字符串中的内容,迭代器start和end指示范围,

- 用num个ch字符替换本字符串中的内容，迭代器start和end指示范围.

例如，以下代码显示字符串"They say he carved it himself...find your soul-mate, Homer."

```
string s = "They say he carved it himself...from a BIGGER spoon";
string s2 = "find your soul-mate, Homer.";

s.replace( 32, s2.length(), s2 );

cout << s << endl;
```

*相关主题:*
**insert()**

---

## 保留空间(reserve)

*语法:*

```
void reserve( size_type num );
```

reserve()函数设置本字符串的capacity 以保留num个字符空间。

*相关主题:*
**capacity()**

---

## resize

*语法:*

```
void resize( size_type num );
void resize( size_type num, char ch );
```

resize()函数改变本字符串的大小到*num*，新空间的内容不确定。也可以指定用ch填充。

---

## rfind

*语法:*

```
size_type rfind( const basic_string &str, size_type index );
size_type rfind( const char *str, size_type index );
size_type rfind( const char *str, size_type index, size_type num );
size_type rfind( char ch, size_type index );
```

rfind()函数:

- 返回最后一个与str中的某个字符匹配的字符，从index开始查找。如果没找到就返回**string::npos**
- 返回最后一个与str中的某个字符匹配的字符，从index开始查找,最多查找num个字符。如果没找到就返回**string::npos**
- 返回最后一个与ch匹配的字符，从index开始查找。如果没找到就返回**string::npos**

例如，在下列代码中第一次调用rfind()返回**string::npos**,因为目标词语不在开始的8个字符中。然而，第二次调用返回9，因为目标词语在开始的20个字符之中。

```
int loc;
string s = "My cat's breath smells like cat food.";

loc = s.rfind( "breath", 8 );
cout << "The word breath is at index " << loc << endl;

loc = s.rfind( "breath", 20 );
cout << "The word breath is at index " << loc << endl;
```

*相关主题:*
**find()**

## size

*语法:*

```
size_type size();
```

size()函数返回字符串中现在拥有的字符数。

*相关主题:*
**length(), max_size()**

## substr

*语法:*

```
basic_string substr( size_type index, size_type num = npos );
```

substr()返回本字符串的一个子串，从index开始，长num个字符。如果没有指定，将是默认值 **string::npos**。这样，substr()函数将简单的返回从index开始的剩余的字符串。

例如:

```
string s("What we have here is a failure to communicate");

string sub = s.substr(21);

cout << "The original string is " << s << endl;
cout << "The substring is " << sub << endl;
```

显示:

```
The original string is What we have here is a failure to communicate
The substring is a failure to communicate
```

## 交换(swap)

*语法:*

```
void swap( basic_string &str );
```

swap()函数把str和本字符串交换。例如:

```
string first( "This comes first" );
string second( "And this is second" );
first.swap( second );
cout << first << endl;
cout << second << endl;
```

显示:

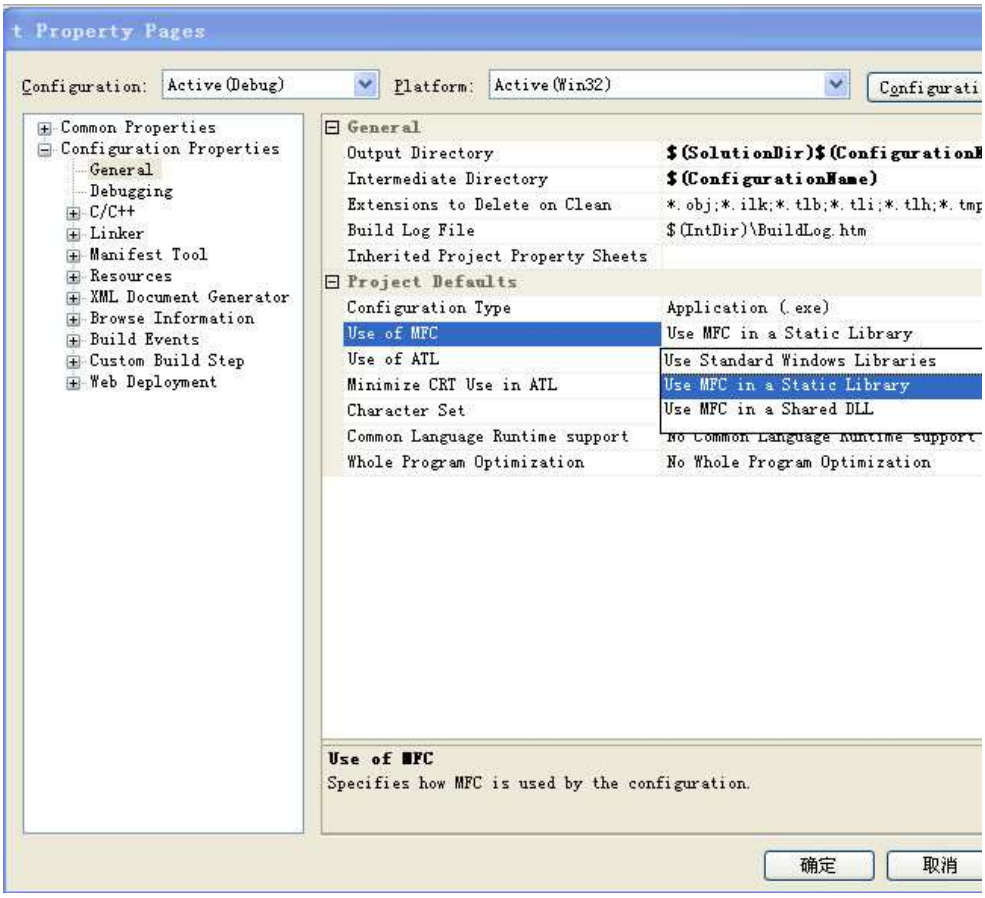
```
And this is second
This comes first
```

以上就是对C++ string类的一个简要介绍。用的好的话它所具有的功能不会比MFC中的CString类逊色多少，呵呵，个人意见!

最后要介绍如何在Win32 应用程序中引用MFC中的部分类，例如CString。

1.在工程目录下右键选择"Properties"--->"Configuration Properties"--->"General"--->"Use of MFC"--->"Use MFC in a Static Library",

默认的是："Use Standard Windows Libraries"，如下图:




2.在你所用的所有头文件之前包含#include <afxwin.h>，例如：可以在stdafx.h文件的最前面包含#include <afxwin.h>头文件，这样在你的源代码中就可以使用

CString类了，不过这样也有一个缺点，就是编译出来的程序要比原来的大很多。我试过一个小程序，选择"Use Standard Windows Libraries" 编译出来

的Release版本大概92kb，使用"Use MFC in a Static Library"编译出来的Release版本大概192kb，足足大了100kb，这个就个人考虑了.....

分类: c++



aminxu
关注 - 7
粉丝 - 10
+加关注

2 0

« 上一篇: C++中数字与字符串之间的转换（转）
» 下一篇: C++中vector使用详细说明 (转)

posted @ 2015-07-29 16:37 aminxu 阅读(55786) 评论(0) 编辑 收藏

**相关博文：**
· 标准C++中的string类的用法总结
· 标准C++中的string类的用法总结
· 标准C++中的string类的用法总结
· 标准C++中string类的用法总结
· 标准C++中的string类的用法总结