

-
-
-
-
-
-
-
-
-

- [网志](#)
- [列表](#)
- [标签](#)
- [项目](#)
- [游记](#)
- [关于](#)
- [联系](#)

十月212008

- 作者:BYVoid
- - [計算機科學](#)
 - [C++](#)
 - [技術](#)
 - [字符串](#)
 - [string](#)
 - [用法](#)
- 阅读: 232789

C++ string 用法详解

C++ string 详解

任何人对本文进行引用都要标明作者是Nicolai M.Josuttis

////////////////////////////////////

C++ 语言是个十分优秀的语言，但优秀并不表示完美。还是有许多人不愿意使用C或者C++，为什么？原因众多，其中之一就是C/C++的文本处理功能太麻烦，用起来很不方便。以前没有接触过其他语言时，每当别人这么说，我总是不屑一顾，认为他们根本就没有领会C++的精华，或者不太懂C++，现在我接触 perl, php, 和Shell脚本以后，开始理解了以前为什么有人说C++文本处理不方便了。

举例来说，如果文本格式是：用户名 电话号码，文件名name.txt Tom 23245332 Jenny 22231231 Heny 22183942 Tom 23245332 ...

现在我们需要对用户名排序，且只输出不同的姓名。

那么在shell 编程中，可以这样用：

```
awk '{print $1}' name.txt | sort | uniq
```

简单吧？

如果使用C/C++ 就麻烦了，他需要做以下工作：先打开文件，检测文件是否打开，如果失败，则退出。声明一个足够大得二维字符数组或者一个字符指针数组 读入一行到字符空间 然后分析一行的结构，找到空格，存入字符数组中。关闭文件 写一个排序函数，或者使用写一个比较函数，使用sort()排序 遍历数组，比较是否有相同的，如果有，则要删除，copy... 输出信息

你可以用C++或者C语言去实现这个流程。如果一个人的主要工作就是处理这种类似的文本(例如做apache的日志统计和分析),你说他会喜欢C/C++么？

当然，有了STL，这些处理会得到很大的简化。我们可以使用 fstream来代替麻烦的fopen fread fclose, 用vector来代替数组。最重要的是用 string来代替char * 数组，使用sort排序算法来排序，用unique 函数来去重。听起来好像很不错。看看下面代码(例程1)：

```
#include <string>
#include <iostream>
#include <algorithm>
#include <vector>
#include <fstream>
using namespace std;
int main()
{
    ifstream in("name.txt");
    string strtmp;
    vector<string> vect;
    while(getline(in, strtmp, '\n'))
        vect.push_back(strtmp.substr(0, strtmp.find(' ')));
    sort(vect.begin(), vect.end());
    vector<string>::iterator it=unique(vect.begin(), vect.end());
    copy(vect.begin(), it, ostream_iterator<string>(cout, "\n"));
    return 0;
}
```

也还不错吧，至少会比想象得要简单得多！（代码里面没有对错误进行处理，只是为了说明问题，不要效仿）。

当然，在这个文本格式中，不用vector而使用map会更有扩充性，例如，还可通过人名找电话号码等等，但是使用了map就不那么好用sort了。你可以用map试一试。

这里string的作用不只是可以存储字符串，还可以提供字符串的比较，查找等。在sort和unique函数中就默认使用了less 和equal_to函数，上面的一段代码，其实使用了string的以下功能： 存储功能，在getline() 函数中 查找功能，在find() 函数中 子串功能，在substr() 函数中 string operator <，默认在sort() 函数中调用 string operator ==，默认在unique() 函数中调用

总之，有了string 后，C++的字符文本处理功能总算得到了一定补充，加上配合STL其他容器使用，其在文本处理上的功能已经与perl, shell, php的距离缩小很多了。因此掌握string 会让你工作事半功倍。

1 string 使用

其实，string并不是一个单独的容器，只是basic_string 模板类的一个typedef 而已，相对应的还有wstring，你在string 头文件中你会发现下面的代码：

```
extern "C++" {
typedef basic_string<char> string;
typedef basic_string<wchar_t> wstring;
} // extern "C++"
```

由于只是解释string的用法，如果没有特殊的说明，本文并不区分string 和 basic_string的区别。

string 其实相当于一个保存字符的序列容器，因此除了有字符串的一些常用操作以外，还有包含了所有的序列容器的操作。字符串的常用操作包括：增加、删除、修改、查找比较、链接、输入、输出等。详细函数列表参看附录。不要害怕这么多函数，其实有许多是序列容器带有的，平时不一定用的上。

如果你要想了解所有函数的详细用法，你需要查看basic_string，或者下载STL编程手册。这里通过实例介绍一些常用函数。

1.1 充分使用string 操作符

string 重载了许多操作符，包括 +, +=, <, =, [], <<, >>等，正式这些操作符，对字符串操作非常方便。先看看下面这个例子：

```
#include <string>
#include <iostream>
using namespace std;
int main()
{
    string strinfo="Please input your name:";
    cout << strinfo ;
    cin >> strinfo;
    if( strinfo == "winter" )
        cout << "you are winter!"<<endl;
    else if( strinfo != "wende" )
        cout << "you are not wende!"<<endl;
    else if( strinfo < "winter")
        cout << "your name should be ahead of winter"<<endl;
    else
        cout << "your name should be after of winter"<<endl;
    strinfo += " , Welcome to China!";
    cout << strinfo<<endl;
    cout <<"Your name is :"<<endl;
    string strtmp = "How are you? " + strinfo;
    for(int i = 0 ; i < strtmp.size(); i ++ )
        cout<<strtmp[i];
    return 0;
}
```

下面是程序的输出

```
Please input your name:Hero
you are not wende!
Hero , Welcome to China!
How are you? Hero , Welcome to China!
```

有了这些操作符，在STL中仿函数都可以直接使用string作为参数，例如 less, great, equal_to 等，因此在把string作为参数传递的时候，它的使用和int 或者float等已经没有什么区别了。例如，你可以使用：

```
map<string, int> mymap; //以上默认使用了 less<string>
```

有了 operator + 以后，你可以直接连加，例如：

```
string strinfo="winter";
string strlast="Hello " + strinfo + "!";
string strtest="Hello " + strinfo + " Welcome" + " to China" + " !";//你还可以这样：
```

看见其中的特点了吗？只要你的等式里面有一个 string 对象，你就可以一直连续"+，但有一点需要保证的是，在开始的两项中，必须有一项是 string 对象。其原理很简单：

系统遇到"+"号，发现有一项是string 对象。系统把另一项转化为一个临时 string 对象。执行 operator + 操作，返回新的临时string 对象。如果又发现"+"号，继续第一步操作。

由于这个等式是由左到右开始检测执行，如果开始两项都是const char，程序自己并没有定义两个const char 的加法，编译的时候肯定就有问题了。

有了操作符以后，assign(), append(), compare(), at()等函数，除非有一些特殊的需求时，一般是用不上。当然at()函数还有一个功能，那就是检查下标是否合法，如果是使用：

```
string str="winter";//下面一行有可能会引起程序中断错误
str[100]='!';//下面会抛出异常:throws: out_of_range
cout<<str.at(100)<<endl;
```

了解了吗？如果你希望效率高，还是使用[]来访问，如果你希望稳定性好，最好使用at()来访问。

1.2 眼花缭乱的string find 函数

由于查找是使用最为频繁的功能之一，string 提供了非常丰富的查找函数。其列表如下：

函数名	描述
find	查找
rfind	反向查找
find_first_of	查找包含子串中的任何字符，返回第一个位置
find_first_not_of	查找不包含子串中的任何字符，返回第一个位置
find_last_of	查找包含子串中的任何字符，返回最后一个位置
find_last_not_of	查找不包含子串中的任何字符，返回最后一个位置

以上函数都是被重载了4次，以下是以find_first_of 函数为例说明他们的参数，其他函数和其参数一样，也就是说总共有24个函数：

```
size_type find_first_of(const basic_string& s, size_type pos = 0)
size_type find_first_of(const charT* s, size_type pos, size_type n)
size_type find_first_of(const charT* s, size_type pos = 0)
size_type find_first_of(charT c, size_type pos = 0)
```

所有的查找函数都返回一个size_type类型，这个返回值一般都是所找到字符串的位置，如果没有找到，则返回string::npos。有一点需要特别注意，所有和string::npos的比较一定要用string::size_type来使用，不要直接使用int 或者unsigned int等类型。其实string::npos表示的是-1, 看看头文件：

```
template <class _CharT, class _Traits, class _Alloc>
const basic_string<_CharT, _Traits, _Alloc>::size_type
basic_string<_CharT, _Traits, _Alloc>::npos
= basic_string<_CharT, _Traits, _Alloc>::size_type) -1;
```

find 和 rfind 都还比较容易理解，一个是正向匹配，一个是逆向匹配，后面的参数pos都是用来指定起始查找位置。对于find_first_of 和find_last_of 就不是那么好理解。

find_first_of 是给定一个要查找的字符集，找到这个字符集中任何一个字符所在字符串中第一个位置。或许看一个例子更容易明白。

有这样一个需求：过滤一行开头和结尾的所有非英文字符。看看用string 如何实现：

```
#include <string>
#include <iostream>
using namespace std;
int main()
{
    string strinfo= " /*---Hello Word!.....-----";
    string strset="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
    int first = strinfo.find_first_of(strset);
    if(first == string::npos)
    {
        cout<<"not find any characters"<<endl;
        return -1;
    }
    int last = strinfo.find_last_of(strset);
    if(last == string::npos)
    {
        cout<<"not find any characters"<<endl;
        return -1;
    }
    cout << strinfo.substr(first, last - first + 1)<<endl;
    return 0;
}
```

这里把所有的英文字母大小写作为需要查找的字符集，先查找第一个英文字母的位置，然后查找最后一个英文字母的位置，然后用substr 来的到中间的一部分，用于输出结果。下面就是其结果：

Hello Word

前面的符号和后面的符号都没有了。像这种用法可以用来查找分隔符，从而把一个连续的字符串分割成为几部分，达到 shell 命令中的 awk 的用法。特别是当分隔符有多个的时候，可以一次指定。例如有这样的需求：

```
张三|3456123, 湖南
李四,4564234| 湖北
王小二, 4433253|北京
...
```

我们需要以 "|" ", "为分隔符，同时又要过滤空格，把每行分成相应的字段。可以作为你的一个作业来试试，要求代码简洁。

1.3 string insert, replace, erase

了解了string 的操作符，查找函数和substr，其实就已经了解了string的80%的操作了。insert函数, replace函数和erase函数在使用起来相对简单。下面以一个例子来说明其应用。string只是提供了按照位置和区间的replace函数，而不能用一个string字符串来替换指定string中的另一个字串。这里写一个函数来实现这个功能：

```
void string_replace(string & strBig, const string & strsrc, const string & strdst)
{
    string::size_type pos=0;
    string::size_type srclen=strsrc.size();
    string::size_type dstlen=strdst.size();
    while( (pos=strBig.find(strsrc, pos)) != string::npos)
    {
        strBig.replace(pos, srclen, strdst);
        pos += dstlen;
    }
}
```

看看如何调用：

```
#include <string>
#include <iostream>
using namespace std;
int main()
{
    string strinfo="This is Winter, Winter is a programmer. Do you know Winter?";
    cout<<"Orign string is :\\n"<<strinfo<<endl;
    string_replace(strinfo, "Winter", "wende");
    cout<<"After replace Winter with wende, the string is :\\n"<<strinfo<<endl;
    return 0;
}
```

其输出结果：

```
Orign string is :
This is Winter, Winter is a programmer. Do you know Winter?
After replace Winter with wende, the string is :
This is wende, wende is a programmer. Do you know wende?
```

如果不用replace函数，则可以使用erase和insert来替换，也能实现string_replace函数的功能：

```
void string_replace(string & strBig, const string & strsrc, const string &strdst)
{
    string::size_type pos=0;
    string::size_type srclen=strsrc.size();
    string::size_type dstlen=strdst.size();
    while( (pos=strBig.find(strsrc, pos)) != string::npos)
    {
        strBig.erase(pos, srclen);
        strBig.insert(pos, strdst);
        pos += dstlen;
    }
}
```

当然，这种方法没有使用replace来得直接。

2 string 和C-style字符串

现在看了这么多例子，发现const char* 可以和string 直接转换，例如我们在上面的例子中，使用

```
string_replace(strinfo, "Winter", "wende");来代用void string_replace(string & strBig, const string & strsrc, const string &strdst)
```

在C语言中只有char 和 const char，为了使用起来方便，string提供了三个函数满足其要求：

```
const charT* c_str() const
const charT* data() const
size_type copy(charT* buf, size_type n, size_type pos = 0) const
```

其中：c_str 直接返回一个以\0结尾的字符串。data 直接以数组方式返回string的内容，其大小为size()的返回值，结尾并没有\0字符。copy 把string的内容拷贝到buf空间中。你或许会问，c_str()的功能包含data()，那还需要data()函数干什么？看看源码：

```
const charT* c_str () const
{
    if (length () == 0)
        return "";
    terminate ();
    return data ();
}
```

原来c_str()的流程是：先调用terminate()，然后在返回data()。因此如果你对效率要求比较高，而且你的处理又不一定需要以\0的方式结束，你最好选择data()。但是对于一般的C函数中，需要以const char*为输入参数，你就要使用c_str()函数。

对于c_str() data()函数，返回的数组都是由string本身拥有，千万不可修改其内容。其原因是许多string实现的时候采用了引用机制，也就是说，有可能几个string使用同一个字符存储空间。而且你不能使用sizeof(string)来查看其大小。详细的解释和实现查看Effective STL的条款15：小心string实现的多样性。

另外在你的程序中，只在需要时才使用c_str()或者data()得到字符串，每调用一次，下次再使用就会失效，如：

```
string strinfo("this is Winter");
...
//最好的方式是：
foo(strinfo.c_str());
//也可以这么用：
const char* pstr=strinfo.c_str();
foo(pstr);
//不要再使用了pstr了，下面的操作已经使pstr无效了。
strinfo += " Hello!";
foo(pstr);//错误！
```

会遇到什么错误？当你幸运的时候pstr可能只是指向"this is Winter Hello!"的字符串，如果不幸运，就会导致程序出现其他问题，总会有一些不可遇见的错误。总之不会是你预期的那个结果。

3 string 和 Charactor Traits 了解了string的用法，该详细看看string的真相了。前面提到string 只是basic_string的一个typedef。看看basic_string 的参数：

```
template <class charT, class traits = char_traits<charT>,
class Allocator = allocator<charT> >
class basic_string
{
```

```
//...
}
```

`char_traits`不仅是在`basic_string` 中有用，在`basic_istream` 和 `basic_ostream`中也需要用到。就像Steve Donovan在过度使用C++模板中提到的，这些确实有些过头了，要不是系统自己定义了相关的一些属性，而且用了个`typedef`，否则还真不知道如何使用。

但复杂总有复杂道理。有了`char_traits`，你可以定义自己的字符串类型。当然，有了`char_traits < char >` 和`char_traits < wchar_t >` 你的需求使用已经足够了，为了更好的理解`string`，咱们来看看`char_traits`都有哪些要求。

如果你希望使用你自己定义的字符，你必须定义包含下列成员的结构： 表达式 描述

```
char_type 字符类型
int_type int 类型
pos_type 位置类型
off_type 表示位置之间距离的类型
state_type 表示状态的类型
assign(c1,c2) 把字符c2赋值给c1
eq(c1,c2) 判断c1,c2 是否相等
lt(c1,c2) 判断c1是否小于c2
length(str) 判断str的长度
compare(s1,s2,n) 比较s1和s2的前n个字符
copy(s1,s2, n) 把s2的前n个字符拷贝到s1中
move(s1,s2, n) 把s2中的前n个字符移动到s1中
assign(s,n,c) 把s中的前n个字符赋值为c
find(s,n,c) 在s的前n个字符内查找c
eof() 返回end-of-file
to_int_type(c) 将c转换成int_type
to_char_type(i) 将i转换成char_type
not_eof(i) 判断i是否为EOF
eq_int_type(i1,i2) 判断i1和i2是否相等
```

想看看实际的例子，你可以看看`sgi STL`的`char_traits`结构源码。

现在默认的`string`版本中，并不支持忽略大小写的比较函数和查找函数，如果你想练练手，你可以试试改写一个`char_traits`，然后生成一个`case_string`类，也可以在`string` 上做继承，然后派生一个新的类，例如：`ext_string`，提供一些常用的功能，例如：

定义分隔符。给定分隔符，把`string`分为几个字段。提供替换功能。例如，用`winter`, 替换字符串中的`wende` 大小写处理。例如，忽略大小写比较，转换等 整形转换。例如把"123"字符串转换为123数字。这些都是常用的功能，如果你有兴趣可以试试。其实有人已经实现了，看看`Extended STL string`。如果你想偷懒，下载一个头文件就可以用，有了它确实方便了很多。要是有人能提供一个支持正则表达式的`string`，我会非常乐意用。

4 `string` 建议 使用`string` 的方便性就不用再说了，这里要重点强调的是`string`的安全性。`string`并不是万能的，如果你在一个大工程中需要频繁处理字符串，而且有可能是多线程，那么你一定要慎重(当然，在多线程下你使用任何`STL`容器都要慎重)。`string`的实现和效率并不一定是你想象的那样，如果你对大量的字符串操作，而且特别关心其效率，那么你有两个选择，首先，你可以看看你使用的`STL`版本中`string`实现的源码；另一选择是你自己写一个只提供你需要的功能的类。`string`的`c_str()`函数是用来得到C语言风格的字符串，其返回的指针不能修改其空间。而且在下一次使用时重新调用获得新的指针。`string`的`data()`函数返回的字符串指针不会以'\0'结束，千万不可忽视。尽量去使用操作符，这样可以让程序更加易懂

5 小结 难怪有人说：`string` 使用方便功能强，我们一直用它！

6 附录

```
string 函数列表 函数名 描述
begin 得到指向字符串开头的Iterator
end 得到指向字符串结尾的Iterator
rbegin 得到指向反向字符串开头的Iterator
rend 得到指向反向字符串结尾的Iterator
size 得到字符串的大小
length 和size函数功能相同
max_size 字符串可能的最大大小
capacity 在不重新分配内存的情况下，字符串可能的大小
empty 判断是否为空
operator[] 取第几个元素，相当于数组
c_str 取得C风格的const char* 字符串
data 取得字符串内容地址
operator= 赋值操作符
reserve 预留空间
swap 交换函数
insert 插入字符
append 追加字符
push_back 追加字符
operator+= += 操作符
erase 删除字符串
clear 清空字符容器中所有内容
resize 重新分配空间
assign 和赋值操作符一样
replace 替代
copy 字符串到空间
find 查找
rfind 反向查找
find_first_of 查找包含子串中的任何字符，返回第一个位置
find_first_not_of 查找不包含子串中的任何字符，返回第一个位置
find_last_of 查找包含子串中的任何字符，返回最后一个位置
find_last_not_of 查找不包含子串中的任何字符，返回最后一个位置
```

```

substr 得到子串
compare 比较字符串
operator+ 字符串链接
operator== 判断是否相等
operator!= 判断是否不等于
operator< 判断是否小于
operator>> 从输入流中读入字符串
operator<< 字符串写入输出流
getline 从输入流中读入一行

```

```

////////////////////////////////////

```

之所以抛弃char*的字符串而选用C++标准程序库中的string类，是因为他和前者比较起来，不必担心内存是否足够、字符串长度等等，而且作为一个类出现，他集成的操作函数足以完成我们大多数情况下(甚至是100%)的需要。我们可以用 = 进行赋值操作，== 进行比较，+ 做串联（是不是很简单?）。我们尽可以把它看成是C++的基本数据类型。好了，进入正题..... 首先，为了在我们的程序中使用string类型，我们必须包含头文件<string>。如下：

```

#include <string> //注意这里不是string.h string.h是C字符串头文件

```

1. 声明一个C++字符串 声明一个字符串变量很简单：

```

string Str;

```

这样我们就声明了一个字符串变量，但既然是一个类，就有构造函数和析构函数。上面的声明没有传入参数，所以就直接使用了string的默认的构造函数，这个函数所作的就是把Str初始化为一个空字符串。String类的构造函数和析构函数如下：

```

a)   string s; //生成一个空字符串s
b)   string s(str) //拷贝构造函数 生成str的复制品
c)   string s(str,stridx) //将字符串str内“始于位置stridx”的部分当作字符串的初值
d)   string s(str,stridx,strlen) //将字符串str内“始于stridx且长度顶多strlen”的部分作为字符串的初值
e)   string s(cstr) //将C字符串作为s的初值
f)   string s(chars,chars_len) //将C字符串前chars_len个字符作为字符串s的初值。
g)   string s(num,c) //生成一个字符串，包含num个c字符
h)   string s(beg,end) //以区间beg;end(不包含end)内的字符作为字符串s的初值
i)   s.~string() //销毁所有字符，释放内存

```

都很简单，我就不解释了。

2. 字符串操作函数 这里是C++字符串的重点，我先把各种操作函数罗列出来，不喜欢把所有函数都看完的人可以在这里找自己喜欢的函数，再到后面看他的详细解释。

```

a) =,assign() //赋以新值
b) swap() //交换两个字符串的内容
c) +=,append(),push_back() //在尾部添加字符
d) insert() //插入字符
e) erase() //删除字符
f) clear() //删除全部字符
g) replace() //替换字符
h) + //串联字符串
i) ==,!=,<,<=,>,>=,compare() //比较字符串
j) size(),length() //返回字符数量
k) max_size() //返回字符的可能最大个数
l) empty() //判断字符串是否为空
m) capacity() //返回重新分配之前的字符容量
n) reserve() //保留一定量内存以容纳一定数量的字符
o) [ ], at() //存取单一字符
p) >>,getline() //从stream读取某值
q) << //将某值写入stream
r) copy() //将某值赋值为一个C_string
s) c_str() //将内容以C_string返回
t) data() //将内容以字符数组形式返回
u) substr() //返回某个子字符串
v) 查找函数
w)begin() end() //提供类似STL的迭代器支持
x) rbegin() rend() //逆向迭代器
y) get_allocator() //返回配置器

```

下面详细介绍：

2. 1 C++字符串和C字符串的转换

C++提供的由C++字符串得到对应的C_string的方法是使用data()、c_str()和copy()，其中，data()以字符数组的形式返回字符串内容，但并不添加'\0'。c_str()返回一个以'\0'结尾的字符串。

2. 2 大小和容量函数 一个C++字符串存在三种大小：a)现有的字符数，函数是size()和length()，他们等效。Empty()用来检查字符串是否为空。b)max_size() 这个大小是指当前C++字符串最多能包含的字符数，很可能和机器本身的限制或者字符串所在位置连续内存的大小有关系。我们一般情况下不用关心他，应该大小足够我们用的。但是不够用的话，会抛出length_error异常c)capacity()重新分配内存之前 string所能包含的最大字符数。这里另一个需要指出的是reserve()函数，这个函数为string重新分配内存。重新分配的大小由其参数决定，默认参数为0，这时候会对string进行非强制性缩减。

还有必要再重复一下C++字符串和C字符串转换的问题，许多人会遇到这样的问题，自己做的程序要调用别人的函数、类什么的（比如数据库连接函数Connect(char,char)），但别人的函数参数用的是char形式的，而我们知道，c_str()、data()返回的字符数组由该字符串拥有，所以是一种const char,要想作为上面提及的函数的参数，还必须拷贝到一个char,而我们的原则是能不使用C字符串就不使用。那么，这时候我们的处理方式是：如果此函数对参数(也就是char)的内容不修改的话，我们可以这样Connect((char)UserID.c_str(), (char)PassWD.c_str()),但是这时候是存在危险的，因为这样转换后的字符

串其实是修改的（有兴趣可以自己试一试），所以我强调除非函数调用的时候不对参数进行修改，否则必须拷贝到一个char上去。当然，更稳妥的办法是无论什么情况都拷贝到一个char上去。同时我们也祈祷现在仍然使用C字符串进行编程的高手们（说他们是高手一点儿也不为过，也许在我们还穿开裆裤的时候他们就开始编程了，哈哈...）写的函数都比较规范，那样我们就不必进行强制转换了。

2. 3元素存取 我们可以使用下标操作符[]和函数at()对元素包含的字符进行访问。但是应该注意的是操作符[]并不检查索引是否有效（有效索引0~str.length()），如果索引失效，会引起未定义的行为。而at()会检查，如果使用at()的时候索引无效，会抛出 out_of_range异常。有一个例外不得不说，const string a;的操作符[]对索引值是a.length()仍然有效，其返回值是'\0'。其他的各种情况，a.length()索引都是无效的。举例如下：

```
const string Cstr("const string");
string Str("string");

Str[3];    //ok
Str.at(3); //ok

Str[100]; //未定义的行为
Str.at(100); //throw out_of_range

Str[Str.length()] //未定义行为
Cstr[Cstr.length()] //返回 '\0'
Str.at(Str.length()); //throw out_of_range
Cstr.at(Cstr.length()) ///throw out_of_range
```

我不赞成类似于下面的引用或指针赋值：char& r=s[2]; char* p= &s[3];

因为一旦发生重新分配，r,p立即失效。避免的方法就是不使用。

2. 4比较函数 C++字符串支持常见的比较操作符(>,>=,<,<=,==,!=)，甚至支持string与C-string的比较(如str<"hello")。在使用>,>=,<,<=这些操作符的时候是根据“当前字符特性”将字符按字典顺序进行逐一得比较。字典排序靠前的字符小，比较的顺序是从前向后比较，遇到不相等的字符就按这个位置上的两个字符的比较结果确定两个字符串的大小。同时，string("aaaa")<string(aaaaa)。另一个功能强大的比较函数是成员函数compare()。他支持多参数处理，支持用索引值和长度定位子串来进行比较。他返回一个整数来表示比较结果，返回值意义如下：0-相等 > 0-大于 < 0-小于。举例如下：

```
string s("abcd");
s.compare("abcd"); //返回0
s.compare("dcba"); //返回一个小于0的值
s.compare("ab"); //返回大于0的值
s.compare(s); //相等
s.compare(0,2,s,2,2); //用"ab"和"cd"进行比较 小于零
s.compare(1,2,"bcx",2); //用"bc"和"bc"比较。
```

怎么样？功能够全的吧！什么？还不能满足你的胃口？好吧，那等着，后面有更个性化的比较算法。先给个提示，使用的是STL的比较算法。什么？对STL一窍不通？你重修吧！

2. 5 更改内容 这在字符串的操作中占了很大一部分。首先讲赋值，第一个赋值方法当然是使用操作符=，新值可以是string(如：s=ns)、c_string(如：s="gaint")甚至单一字符（如：s='j'）。还可以使用成员函数assign()，这个成员函数可以使你更灵活的对字符串赋值。还是举例说明吧：

```
s.assign(str); //直接
s.assign(str,1,3); //如果str是"iamangel" 就是把"ama"赋给字符串
s.assign(str,2,string::npos); //把字符串str从索引值2开始到结尾赋给s
s.assign("gaint"); //不说
s.assign("nico",5); //把'n' 'I' 'c' 'o' '\0'赋给字符串
s.assign(5,'x'); //把五个x赋给字符串
```

把字符串清空的方法有三个：s="" ;s.clear();s.erase();(我越来越觉得举例比说话让别人容易懂！)。string提供了很多函数用于插入(insert)、删除(erase)、替换(replace)、增加字符。先说增加字符（这里说的增加是在尾巴上），函数有+=、append()、push_back()。举例如下：

```
s+=str; //加个字符串
s+="my name is jiayp"; //加个C字符串
s+='a'; //加个字符

s.append(str);
s.append(str,1,3); //不解释了 同前面的函数参数assign的解释
s.append(str,2,string::npos) //不解释了

s.append("my name is jiayp");
s.append("nico",5);
s.append(5,'x');

s.push_back('a'); //这个函数只能增加单个字符 对STL熟悉的理解起来很简单
```

也许你需要在string中间的某个位置插入字符串，这时候你可以用insert()函数，这个函数需要你指定一个安插位置的索引，被插入的字符串将放在这个索引的后面。

```
s.insert(0,"my name");
s.insert(1,str);
```

这种形式的insert()函数不支持传入单个字符，这时的单个字符必须写成字符串形式(让人恶心)。既然你觉得恶心，那就不得不继续读下面一段话：为了插入单个字符，insert()函数提供了两个对插入单个字符操作的重载函数：insert(size_type index,size_type num,chart c)和insert(iterator pos,size_type num,chart c)。其中size_type是无符号整数，iterator是char*，所以，你这么调用insert函数是不行的：insert(0,1,'j');这时候第一个参数将转换成哪一个呢？所以你必须这么写：insert((string::size_type)0,1,'j')！第二种形式指出了使用迭代器安插字符的形式，在后面会提及。顺便提一下，string有很多操作是使用STL的迭代器的，他也尽量做得和STL靠近。删除函数erase()的形式也有好几种（真烦！），替换函数replace()也有好几个。举例吧：

```
string s="il8n";
s.replace(1,2,"nternationalizatio"); //从索引1开始的2个替换成后面的C_string
```

```
s.erase(13); //从索引13开始往后全删除
s.erase(7,5); //从索引7开始往后删5个
```

2. 6提取子串和字符串连接

提取子串的函数是：`substr()`，形式如下：

```
s.substr(); //返回s的全部内容
s.substr(11); //从索引11往后的子串
s.substr(5,6); //从索引5开始6个字符
```

把两个字符串结合起来的函数是`+`。（谁不明白请致电120）

2. 7输入输出操作 1. `>>` 从输入流读取一个string。 2. `<<` 把一个string写入输出流。 另一个函数就是`getline()`，他从输入流读取一行内容，直到遇到分行符或到了文件尾。

2. 8搜索与查找 查找函数很多，功能也很强大，包括了：

```
find()
rfind()
find_first_of()
find_last_of()
find_first_not_of()
find_last_not_of()
```

这些函数返回符合搜索条件的字符区间内的第一个字符的索引，没找到目标就返回`npos`。所有的函数的参数说明如下： 第一个参数是被搜寻的对象。第二个参数（可有可无）指出string内的搜寻起点索引，第三个参数（可有可无）指出搜寻的字符个数。比较简单，不多说 不理解的可以向我提出，我再仔细的解答。当然，更加强大的STL搜寻在后面会有提及。最后再说说`npos`的含义，`string::npos`的类型是`string::size_type`，所以，一旦需要把一个索引与`npos`相比，这个索引值必须是`string::size_type`类型的，更多的情况下，我们可以直接把函数和`npos`进行比较（如：`if(s.find("jia")==string::npos)`）。

任何人对本文进行引用都要标明作者是Nicolai M. Josuttis 译者是侯捷/孟岩

[微信扫一扫：分享](#)



[微信里点“发现”，扫一下](#)

[二维码便可将本文分享至朋友圈。](#)

相关日志

- [三种线性排序算法 计数排序、桶排序与基数排序](#) - 42668
- [POI 2000 Repetitions 最长公共子串](#) - 7425
- [USACO 5.5.2 Hidden Passwords 隐藏口令 hidden](#) - 6072
- [Treap](#) - 10791
- [Ubuntu 卸载桌面](#) - 10010
- [几种浏览器的内核](#) - 17539
- [USACO 4.3.4 Letter Game 字母游戏 lgame](#) - 5725
- [高精度计算对象](#) - 6349
- [C++中fstream的用法](#) - 74470
- [C语言字符串函数大全](#) - 55946

15 Comments BYVoid

Login ▾

Recommend 10

Share

Sort by Best ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS

Name



Guest · 4 years ago

多谢如此详细的讲述。

其中这一段

//也可以这么用：

```
const char* pstr=strinfo.c_str();
```

```
foo(pstr);
```

//不要再使用了pstr了。下面的操作已经使pstr失效了。

换言之,我是否可以理解为,有一种设置,使得它被调用后,然后就释放掉了它的内存了.
如果我自己想这么做,能否做个类似的实现呢?
(调用函数申请一个空间,监控被调用情况,一旦被调用结束,就释放它)?

6 ^ | v · Reply · Share ›



blue · 5 years ago

很好

1 ^ | v · Reply · Share ›



Bian · 3 years ago

#include<iterator>

^ | v · Reply · Share ›



Bian → **Bian** · 3 years ago

第一个例子要包含这个头文件

^ | v · Reply · Share ›



映山 邓 · 4 years ago

赞一个

^ | v · Reply · Share ›



Raysmond · 5 years ago

博主总结得如此详细,看起来又很清晰,收藏了哦。

^ | v · Reply · Share ›



yang3wei · 6 years ago

很赞的文章,看起来毫不费力,感谢!

其实,主要还是想看一下能不能准确的取到操作系统和浏览器~^ ^

^ | v · Reply · Share ›



Indeed · 6 years ago

感谢,字符串一度是我转C++的最大障碍。

^ | v · Reply · Share ›



kingslary · 6 years ago

很精彩的文章,讲解很详细,看起来很轻松,学了很多很多,以后也会经常拜读的!

^ | v · Reply · Share ›



Samuel-jk · 6 years ago

很详细,很好。

^ | v · Reply · Share ›



lanstory · 6 years ago

好东西 收藏了~

^ | v · Reply · Share ›



xxzc · 6 years ago

学到了很多,谢谢。

^ | v · Reply · Share ›



Strayer · 7 years ago

感谢BYD神牛 ...

收藏了...

^ | v · Reply · Share ›



harsystraw · 7 years ago

受教了

^ | v · Reply · Share ›



BYVoid Mod · 8 years ago

void string_replace(string & strBig, const string & strsrc, const string &strdst)很好用

^ | v · Reply · Share ›

ALSO ON BYVOID

摩爾多瓦穿越之旅 (一)

2 comments · 10 months ago

wilm — 写的真棒,有木有 (二)


波納佩島意外之旅 (一)


11 comments · 7 months ago

Lei Wang — 我不工作


瑞士旅居記 (二): 歐洲的十字路口

為什麼美國這麼無聊 (一)

 Subscribe

 Add Disqus to your site

Add Disqus

 Privacy

Search for:

语言

- [正體中文](#)
- [簡體中文](#)
- [English](#)

分类

- [中文與漢語](#)
- [生活點滴](#)
- [稷下學宮](#)
- [精華轉載](#)
- [自娛自樂](#)
- [設計開發](#)
- [點滴發現](#)
- [計算機科學](#)
- [競賽題解](#)
- [競賽歷程](#)
- [世界之旅](#)

最热门

- [C++ string 用法详解](#) – 232789
- [推荐一个神级输入法——Rime](#) – 232215
- [有向图强连通分量的Tarjan算法](#) – 214737
- [避讳借字——「屌」、「禽」、「尿」](#) – 164892
- [为什么美国这么无聊（一）](#) – 106861
- [HTTP协议头部与Keep-Alive模式详解](#) – 95619
- [海外实习面试记](#) – 90135
- [各种字符串Hash函数比较](#) – 85316
- [探寻C++最快的读取文件的方案](#) – 80696
- [『古剑奇谭』剧情梗概](#) – 80190
- [这一年来](#) – 79791
- [图的割点、桥与双连通分支](#) – 76016
- [C++中fstream的用法](#) – 74470
- [关于阿里巴巴面试结果信息泄漏的一点说明](#) – 74063
- [广韵查询系统](#) – 65820
- [匈牙利算法](#) – 65122
- [混淆的概念——“繁体字”、“正体字”、“简体字”、“简化字”之辨析](#) – 58972
- [C语言字符串函数大全](#) – 55946
- [美国之行（二）硅谷与旧金山](#) – 54068
- [“祗”、“祗”、“祗”之辨](#) – 52539
- [C/C++的64位整型](#) – 50654
- [美国之行（一）湾区](#) – 50455
- [Node.js中的child_process及进程通信](#) – 50361
- [Vim 语法高亮与自动缩进](#) – 46629
- [美国之行（四）我对美国的印象](#) – 45430
- [劝君惜取少年时](#) – 44048
- [普通话是胡语吗？](#) – 43936
- [线性规划与网络流24题 解题报告](#) – 43175
- [三种线性排序算法 计数排序、桶排序与基数排序](#) – 42668
- [说说「支那」](#) – 40689

存档

- [2017年十月](#) (1)
- [2017年八月](#) (1)
- [2017年六月](#) (1)
- [2017年五月](#) (2)
- [2017年三月](#) (2)
- [2017年二月](#) (3)
- [2016年十二月](#) (1)
- [2016年十月](#) (1)

- [2016年九月](#) (1)
- [2016年八月](#) (1)
- [2016年六月](#) (1)
- [2016年五月](#) (1)
- [2016年三月](#) (1)
- [2016年二月](#) (1)
- [2015年十月](#) (1)
- [2015年九月](#) (1)
- [2015年八月](#) (3)
- [2015年七月](#) (1)
- [2015年五月](#) (1)
- [2015年四月](#) (1)
- [2014年九月](#) (1)
- [2014年六月](#) (3)
- [2014年五月](#) (1)
- [2014年三月](#) (6)
- [2014年一月](#) (1)
- [2013年十二月](#) (1)
- [2013年十月](#) (1)
- [2013年六月](#) (2)
- [2013年五月](#) (4)
- [2013年四月](#) (4)
- [2013年三月](#) (2)
- [2013年二月](#) (1)
- [2013年一月](#) (2)
- [2012年十二月](#) (1)
- [2012年九月](#) (1)
- [2012年八月](#) (2)
- [2012年七月](#) (3)
- [2012年六月](#) (1)
- [2012年五月](#) (1)
- [2012年四月](#) (2)
- [2012年三月](#) (1)
- [2012年二月](#) (2)
- [2012年一月](#) (8)
- [2011年十二月](#) (13)
- [2011年十一月](#) (3)
- [2011年十月](#) (2)
- [2011年九月](#) (1)
- [2011年八月](#) (3)
- [2011年七月](#) (3)
- [2011年六月](#) (6)
- [2011年五月](#) (4)
- [2011年四月](#) (2)
- [2011年二月](#) (3)
- [2010年十二月](#) (4)
- [2010年十一月](#) (2)
- [2010年十月](#) (3)
- [2010年九月](#) (3)
- [2010年八月](#) (7)
- [2010年六月](#) (5)
- [2010年五月](#) (11)
- [2010年四月](#) (10)
- [2010年三月](#) (12)
- [2010年二月](#) (1)
- [2010年一月](#) (10)
- [2009年十二月](#) (5)
- [2009年十一月](#) (11)
- [2009年十月](#) (13)
- [2009年九月](#) (6)
- [2009年八月](#) (2)
- [2009年七月](#) (9)
- [2009年六月](#) (14)
- [2009年五月](#) (16)
- [2009年四月](#) (28)
- [2009年三月](#) (21)
- [2009年二月](#) (18)
- [2009年一月](#) (6)
- [2008年十二月](#) (22)
- [2008年十一月](#) (21)
- [2008年十月](#) (24)

- [2008年九月](#) (6)
- [2008年八月](#) (12)
- [2008年七月](#) (20)
- [2008年六月](#) (23)
- [2008年四月](#) (29)
- [2008年三月](#) (8)
- [2008年二月](#) (1)
- [2008年一月](#) (6)
- [2007年十二月](#) (3)
- [2007年十一月](#) (22)

Blogroll

- [MaskRay](#)
- [Yuxin's Blog](#)
- [BlahGeek](#)
- [Yangzhe1990's Blog](#)

Blogroll

- [Dang Fan's Blog](#)
- [Swj's Home](#)
- [CS Slayer](#)

Blogroll

- [Typeof.net](#)
- [Henry's Blog](#)
- [polyhedron\(古韻\)](#)
- [開微堂](#)

Blogroll

- [優哉幽齋](#)
- [超越時空](#)
- [獨異誌](#)

[↑](#) Originally designed by Site5 WordPress Themes. BYVoid refactored with Node.js, less, jade and CoffeeScript.