

string类相关接口使用方法

Vect. 已于 2025-04-29 11:43:51 修改



C/C++ 专栏收录该内容

6 篇文章

一、auto 和范围 for

这里先补充两个 C++11 的语法

源码请自取：<https://github.com/WoAiXueXiHa>（这是我的GitHub网址，欢迎大家批评指正）

auto

- 早期的 C/C++ 中** auto 修饰的变量，是具有自动存储器的变量**。而在 C++11 中，赋予了新的含义：** auto 作为一个新的类型指示符来指示编译器， auto 声明的变量必须由编译器在编译时推导而得
- 使用 auto 声明指针类型时， auto 和 auto* 无任何区别，但是声明引用类型时必须加 &
- 当在同一行声明多个变量时，这些变量必须是相同的类型，否则编译器将会报错，因为编译器实际 只对第一个类型进行推导，然后用推导出来的类型定义其他变量
- auto 不能作为函数的参数，可以做返回值，但是建议谨慎使用
- auto 不能直接用来声明数组

```
1 // C++11 auto相关
2
3 int func1() {
4     return 24;
5 }
6
7 // 1.不能做参数
8 //void func2(auto a){}
9
10 // 2.可以做返回值，谨慎使用，降低效率
11 auto func3(){}
12
```

内容来源：csdn.net

作者昵称：Vect

原文链接：<https://blog.csdn.net/ting1fengyu1/article/details/147604807>

作者主页：<https://blog.csdn.net/ting1fengyu1>

```

13 int main() {
14     int a = 10;
15     auto b = a;
16     auto c = 'a';
17     auto d = func1();
18     编译报错C3531 类型包含“auto”的符号必须具有初始值设定项
19     //auto e;
20
21     // typeid用于打印类型名
22     cout << typeid(b).name() << endl;
23     cout << typeid(c).name() << endl;
24     cout << typeid(d).name() << endl;
25
26     int x = 20;
27     // 3.修饰指针变量 auto*和auto没有区别
28     auto* px1 = &x;
29     auto px2 = &x;
30
31     // 4.修饰引用变量 必须加&
32     auto& m = x;
33
34     cout << typeid(px1).name() << endl;
35     cout << typeid(px2).name() << endl;
36     cout << typeid(m).name() << endl;
37
38     auto aa = 0, bb = 1;
39     // 5. 连续声明 列表的类型要一致
40     // 编译报错: C3538 在声明符列表中, auto必须始终推导为同一类型
41     // auto cc = 2, dd = 3.0;
42
43     // 6.auto不能声明数组
44     // 编译报错: C3318 数组不能具有其中包含auto的元素类型
45     // auto arr[10] = { 0 };
46     return 0;
47 }

```

内容来源: csdn.net

作者昵称: Vect

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/147604807>

作者主页: <https://blog.csdn.net/ting1fengyu1>

以下是一些输出结果和报错信息：

```
// 编译报错C3531 类型包含“auto”的符号必须具有初始值设定项
auto e;
```

abc E1593 无法推导“auto”类型(需要初始值设定项)

✖ C3531 “e”: 类型包含“auto”的符号必须具有初始值设定项

```
// C++11 auto相关

int func1() {
    return 24;
}

// 1. 不能做参数
// void func2(auto a){}

// 2. 可以做返回值, 谨慎使用, 降低效率
auto func3(){}

int main() {
    int a = 10;
    auto b = a;
    auto c = 'a';
    auto d = func1();
    //// 编译报错C3531 类型包含“auto”的符号必须具有初始值设定项
    // auto e;
    // typeid用于打印类型名
    cout << typeid(b).name() << endl;
    cout << typeid(c).name() << endl;
    cout << typeid(d).name() << endl;
}
```

auto自动识别类型

```
Microsoft Visual Studio 调试器
int
char
int

D:\CODE\CPP\string\x64\Debug\st
按任意键关闭此窗口...
```

CSDN @Vect.

内容来源: csdn.net

作者昵称: Vect.

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/147604807>

作者主页: <https://blog.csdn.net/ting1fengyu1>

```

29 int x = 20;
30 // 3.修饰指针变量 auto*和auto没有区别
31 auto* px1 = &x;
32 auto px2 = &x;
33

```

搜索(Ctrl+E)		搜索深度: 3
名称	值	
px1	0x0000006c2393f8f4 (20)	
px2	0x0000006c2393f8f4 (20)	
添加要监视的项		

```

int x = 20;
// 3.修饰指针变量 auto*和auto没有区别
auto* px1 = &x;
auto px2 = &x;

```

```

// 4.修饰引用变量 必须加&
auto& m = x;

```

m是x的别名

```

cout << typeid(px1).name() << endl;
cout << typeid(px2).name() << endl;
cout << typeid(m).name() << endl;

```

```

int * __ptr64
int * __ptr64
int

```

D:\CODE\CPP\string\
按任意键关闭此窗口.

CSDN @Vect.

内容来源: csdn.net

作者昵称: Vect.

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/147604807>

作者主页: <https://blog.csdn.net/ting1fengyu1>

```

41 auto aa = 0, bb = 1;
42 // 5.
43 auto cc = 2, dd = 3.0;
44
45 return 0;

```

abc E1594 对于此实体“auto”类型是“double”，但之前默示为“int”

✗ C3538 在声明符列表中，“auto”必须始终推导为同一类型

// 6. auto不能声明数组

```

auto arr[10] = { 0 };

```

abc E1589 “auto”类型不能出现在顶级数组类型中

✗ C3318 “auto [10]”: 数组不能具有其中包含“auto”的元素类型

内容来源: csdn.net

作者昵称: Vect.

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/147604807>

作者主页: <https://blog.csdn.net/ting1fengyu1>

范围 **for**

- 对于一个有范围的集合而言，由程序员来说明循环的范围是多余的，有时候还会容易犯错误。因此 C++11 中引入了基于范围的 for 循环。for 循环后的括号由冒号 : 分为两部分：
第一部分是范围内用于迭代的变量，第二部分则表示被迭代的范围，自动迭代，自动取数据，自动判断结束
- 范围 for 可以作用到数组和容器对象上进行遍历
- 范围 for 的底层是迭代器

```
1
2 // 范围for
3 int main() {
4     int arr[5] = { 0,1,2,3,4 };
5     // C++ 98
6     for (size_t i = 0; i < sizeof(arr) / sizeof(int); i++)
7     {
8         cout << arr[i] << " " ;
9     }
10    cout << endl;
11    // 修改数据
12    for (size_t i = 0; i < sizeof(arr) / sizeof(int); i++)
13    {
14        cout << arr[i] * 2 << " ";
15    }
16
17    // C++ 11
18    cout << "~~~~~";
19    cout << endl;
20    for (auto& e : arr) cout << e << " ";
21    for (auto& e : arr) e *= 2;
22    cout << endl;
23    for (auto& e : arr) cout << e << " ";
24
25    cout << endl;
26
27    string str("hello, world!");
28    for (auto str : str) cout << str;
29
30    return 0;
31
```

内容来源: csdn.net

作者昵称: Vect

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/147604807>

作者主页: <https://blog.csdn.net/ting1fengyu1>

}

AI写代码

输出结果：

```
int arr[5] = { 0,1,2,3,4 };
// C++ 98
for (size_t i = 0; i < sizeof(arr) / sizeof(int); i++)
{
    cout << arr[i] << " ";
}
cout << endl;
// 修改数据
for (size_t i = 0; i < sizeof(arr) / sizeof(int); i++)
{
    cout << arr[i] * 2 << " ";
}

// C++ 11
cout << endl;
cout << "~~~~~";
cout << endl;
for (auto& e : arr) cout << e << " ";
for (auto& e : arr) e *= 2;
cout << endl;
for (auto& e : arr) cout << e << " ";

cout << endl;

string str("hello, world!");
for (auto str : str) cout << str;

return 0;
```

Microsoft Visual Studio 调试

0 1 2 3 4

0 2 4 6 8

~~~~~

0 1 2 3 4

0 2 4 6 8

hello, world!

D:\CODE\CPP\string\x64\Debug\string.exe (进程  
按任意键关闭此窗口...)

内容来源: csdn.net

作者昵称: Vect.

CSDN @Vect.

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/147604807>作者主页: <https://blog.csdn.net/ting1fengyu1>

## 二、string 类的常用接口

### 1. string 类对象的常见构造

| (constructor) 函数名称          | 函数功能                   |
|-----------------------------|------------------------|
| string();                   | 构造空的string类对象，注意不是空字符串 |
| string(const char* s);      | 用 C 字符串来构造 string 类对象  |
| string(size_t n, char c);   | string 类对象中包含n个字符c     |
| string (const string& str); | 拷贝构造函数                 |

```
1 // 1. 拷贝构造之类
2 void TestS1() {
3     // 初始化字符串对象 s0
4     string s0("Initing string.");
5
6     // 创建一个空字符串对象 s1
7     string s1;
8
9     // 创建一个string的空类 s11
10    string s11();
11
12    // 使用 s0 拷贝构造, s2 内容和 s0 完全一样
13    string s2(s0);
14
15    // 从 s0 的位置 5 开始, 创建一个新的字符串 s3, 内容是 s0 从索引 5 开始到末尾的字符串
16    string s3(s0, 5);
17
18    cout << s0 << endl;
19    cout << s1 << endl;
20    // 是一个类 cout<<s11会报错
21    //cout << s11 << endl;
22    cout << s2 << endl;
23    cout << s3 << endl;
24 }
25
```

内容来源: csdn.net  
作者昵称: Vect  
原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/147604807>  
作者主页: <https://blog.csdn.net/ting1fengyu1>



```

26
27 int main() {
28     TestS1();
29
30     return 0;
31 }

```



AI写代码

输出结果：

```

void TestS1() {
    // 初始化字符串对象 s0
    string s0("Initing string.");

    // 创建一个空字符串对象 s1
    string s1;

    // 创建一个string的空类 s11
    string s11();

    // 使用 s0 拷贝构造, s2 内容和 s0 完全一样
    string s2(s0);

    // 从 s0 的位置 5 开始, 创建一个新的字符串 s3, 内容是 s0 从索引 5 开始到末尾的字符串
    string s3(s0, 5);

    cout << s0 << endl;
    cout << s1 << endl;
    // 是一个类 cout<<s11会报错
    //cout << s11 << endl;
    cout << s2 << endl;
    cout << s3 << endl;
}

int main() {
    TestS1();

    return 0;
}

```

Microsoft Visual Studio 调试 × + ▾

Initing string.

Initing string.

ng string.

D:\CODE\CPP\string\x64\Debug\string.exe (进程 2286)

按任意键关闭此窗口 . . .

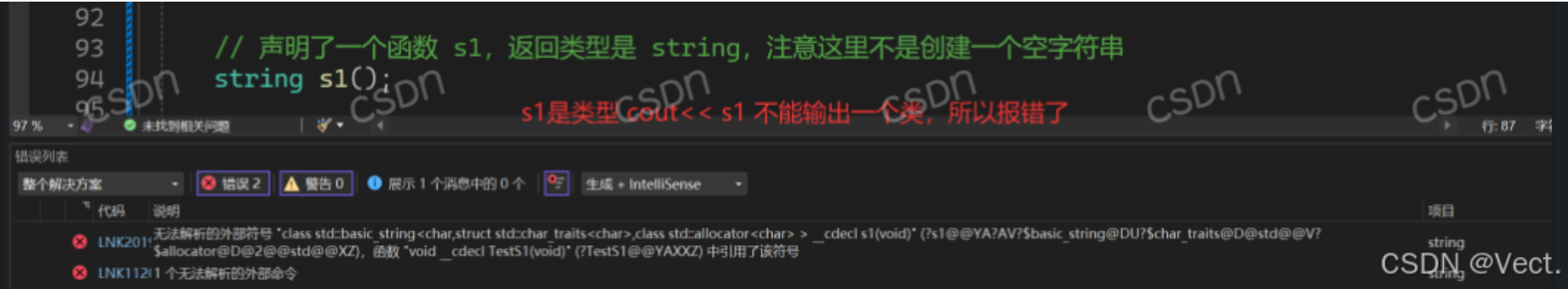
内容来源: csdn.net

作者昵称: Vect.

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/147604807>

作者主页: <https://blog.csdn.net/ting1fengyu1>

这个报错信息要注意：



注意：这里有个非常易错的点，对于 `string()` 官方文档解释为 `Constructs an [empty] string, with a [length of zero characters]`. 可能会有点歧义  
`string s1();` → 声明了一个返回 `string` 的函数 `s1`（不会创建字符串对象）  
`string s1`或 `strings1(" ");` → 正确地调用默认构造函数，创建一个空字符串对象

2. `string` 类对象的访问及遍历操作

| 函数名称                        | 函数功能                                                             |
|-----------------------------|------------------------------------------------------------------|
| <code>operator[]</code>     | 返回 <code>pos</code> 位置的字符， <code>const string</code> 类对象调用       |
| <code>begin+ end</code>     | <code>begin</code> 获取一个字符的迭代器 <code>end</code> 获取最后一个字符下一个位置的迭代器 |
| <code>`rbegin + rend</code> | <code>begin</code> 获取一个字符的迭代器 <code>end</code> 获取最后一个字符下一个位置的迭代器 |
| 范围 <code>for</code>         | C++11 支持更简洁的范围 <code>for</code> 的新遍历方式                           |

```
1 // 遍历操作
2 void TestS2() {
3     string s("hello world");
4     // 1. 下标+[]
5     cout << "operator[]:";
6     for (size_t i = 0; i < s.size(); i++)
7     {
8         cout << s[i];
9     }
10    cout << endl;
11
12 }
```

内容来源: [csdn.net](https://blog.csdn.net/ting1fengyu1)  
作者昵称: Vect.  
原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/147604807>  
作者主页: <https://blog.csdn.net/ting1fengyu1>

```

12 // 2.迭代器begin+end
13 // iterator begin();const_iterator begin() const; 返回迭代器指向的字符串字符的第一个位置
14 // iterator end();const_iterator end() const;      返回迭代器指向的字符串字符最后位置的下一个位置
15 // 迭代器范围[begin,end+1)
16 cout << "迭代器begin+end :";
17
18
19 string::iterator it = s.begin(); // 可以将it理解成指针，但他不是指针
20 while (it != s.end()) {
21     cout << *it;
22     ++it;
23 }
24 cout << endl;
25
26 // 3. 反向迭代器rbegin+rend
27 // string::reverse_iterator rit = s.rbegin();
28 // C++11可以用auto自动识别类型
29 cout << "反向迭代器rbegin+rend :";
30 auto rit = s.rbegin();
31 while (rit != s.rend()) {
32     cout << *rit;
33     ++rit;
34 }
35 cout << endl;
36
37 // 4. 范围for
38 cout << "范围for: ";
39 for (auto ch : s) cout << ch;
40 }

```

AI写代码

内容来源: csdn.net

作者昵称: Vect

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/147604807>

作者主页: <https://blog.csdn.net/ting1fengyu1>

输出结果：

```
// 2. 迭代器begin+end
// iterator begin();const_iterator begin() const; 返回迭代器指向的字符串字符的第一个位置
// iterator end();const_iterator end() const;      返回迭代器指向的字符串字符最后位置的下一个位置
// 迭代器范围[begin,end+1)
cout << "迭代器begin+end :";

string::iterator it = s.begin(); // 可以将it理解成指针
while (it != s.end()) {
    cout << *it;
    ++it;
}
cout << endl;

// 3. 反向迭代器rbegin+rend
// string::reverse_iterator rit = s.rbegin();
// C++11可以用auto自动识别类型
cout << "反向迭代器rbegin+rend :";
auto rit = s.rbegin();
```



3. **string** 类对象的容量操作

| 函数名称       | 函数功能                                        |
|------------|---------------------------------------------|
| size()     | 返回字符串的有效字符数                                 |
| length()   | 返回字符串的有效字符数                                 |
| capacity() | 返回字符串的底层空间大小                                |
| clear()    | 清空有效字符（将字符串置为空字符串）但不改变底层空间大小                |
| empty()    | 检测字符串是否为空，为空返回1，非空返回0                       |
| reserve()  | 为字符串预留额外的n个字符的空间，若n小于capacity，则不改变底层空间      |
| resize()   | 将有效字符串改成n个，两种扩充方式，额外的用指定字符 c 或 \0 填充，一种缩减方式 |

```
1 // 3. 容量操作
2 void TestS3() {
3     // size length capacity resize clear
4     string s0("hello, cs!");
5 }
```

内容来源: [csdn.net](https://blog.csdn.net/ting1fengyu1)  
作者昵称: Vect.  
原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/147604807>  
作者主页: <https://blog.csdn.net/ting1fengyu1>

```

6 cout << s0.size() << endl;// 计算一个字符串的长度 单位字节
7 cout << s0.length() << endl;// 计算一个字符串的长度 单位字节
8 cout << s0.capacity() << endl; // 返回当前vector所分配的内存空间 单位字节
9
10 s0.clear();// 将s0的有效字符清空, 注意: 不改变底层空间大小, 仅仅清理字符
11 cout << s0 << endl;
12 cout <<"size:" << s0.size() << "      capacity:"<< s0.capacity()<< endl;
13
14 // 将s1的有效字符增加到12, 剩下的用字符'a'填充
15 string s1("I like code to C");
16 s1.resize(18, '++');
17 cout << s1 << endl;
18 cout << "size:" << s1.size() << "      capacity:" << s1.capacity() << endl;
19 // 将s1有效字符增加到20, 剩下的用'\0'填充 "I like code to C++\0\0"
20 s1.resize(20);
21 cout << s1 << endl;
22 cout << "size:" << s1.size() << "      capacity:" << s1.capacity() << endl;
23
24 // 将s1有效字符减少到5
25 s1.resize(5);
26 cout << s1 << endl;
27 cout << "size:" << s1.size() << "      capacity:" << s1.capacity()<< endl;
28
29 // void reserve (size_t n = 0);
30 // 为s2预留额外11个字符的空间, 是一种预分配内存的方式
31 // 这意味着在添加字符时, 如果新字符的数量不超过预留的空间, 就不需要重新分配内存, 从而减少了性能开销
32 // 如果 n 大于当前字符串的容量, 该函数会使容器将其容量增加到 n 个字符 (或更多)
33 string s2("hello");
34 cout << "size:" << s2.size() << "      capacity:" << s2.capacity() << endl;
35 s2.reserve(16);
36 cout << "size:" << s2.size() << "      capacity:" << s2.capacity() << endl;
37 s2 += " C++!";
38 cout << s2 << endl;
39 cout << "size:" << s2.size() << "      capacity:" << s2.capacity() << endl;
40
41 // bool empty() const;
42 // 检测字符串是否为空 (即其长度是否为 0) 空返回1 非空返回0
43 string s3 = "";
44 string s4 = "haha";

```

内容来源: csdn.net

作者昵称: Vect

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/147604807>

作者主页: <https://blog.csdn.net/ting1fengyu1>

```

45     bool ret1 = s3.empty();
46     bool ret2 = s4.empty();
47     cout << "ret1:" << ret1 << "    ret2:" << ret2 << endl;
48 }

```

AI写代码

输出结果:

```

10
10
15

size:0    capacity:15
I like code to C++
size:18    capacity:31
I like code to C++
size:20    capacity:31
I lik
size:5     capacity:31
size:5     capacity:15
size:5     capacity:31
hello C++!
size:10    capacity:31
ret1:1    ret2:0

D:\CODE\CPP\string\x64\Debug\string.exe (进程 12452)已退出, 代码为 0 (0x0)。
按任意键关闭此窗口。

```

内容来源: [csdn.net](https://blog.csdn.net/ting1fengyu1)  
 作者昵称: Vect. CSDN @Vect.

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/147604807>  
 作者主页: <https://blog.csdn.net/ting1fengyu1>

注意:

- 1. `size()` 与 `length()` 方法底层实现原理完全相同，引入 `size()` 的原因是为了与其他容器的接口保持一致，一般情况下基本都是用 `size()`。
- 2. `clear()` 只是将 `string` 中有效字符清空，不改变底层空间大小。
- 3. `resize(size_t n)` 与 `resize(size_t n, char c)` 都是将字符串中有效字符个数改变到 `n` 个，不同的是当字符个数增多时：`resize(n)` 用 `0` 来填充多出的元素空间，`resize(size_t n, char c)` 用字符 `c` 来填充多出的元素空间。注意：`resize` 在改变元素个数时，如果是将元素个数增多，可能会改变底层容量的大小，如果是将元素个数减少，底层空间总大小不变。
- 4. `reserve(size_t res_arg=0)`：为 `string` 预留空间，不改变有效元素个数，当 `reserve` 的参数小于 `string` 的底层空间总大小时，`reserver` 不会改变容量大小。

4. `string` 类对象的修改操作

| 函数名称                    | 函数功能                                                                    |
|-------------------------|-------------------------------------------------------------------------|
| <code>push_back`</code> | 在字符串后尾插字符 <code>c</code>                                                |
| <code>append</code>     | 在字符串结尾追加一个字符串                                                           |
| <code>operator+=</code> | 在字符串结尾追加一个字符串                                                           |
| <code>c_str</code>      | 返回 <code>C</code> 格式的字符串（返回一个指向字符数组的指针，这个字符数组包含结尾的 <code>'\0'</code> ）  |
| <code>find+npos</code>  | 从字符串 <code>pos</code> 位置开始往后找字符 <code>c</code> ，返回该字符在字符串中的位置           |
| <code>rfind</code>      | 从字符串 <code>pos</code> 的位置开始往前找字符 <code>c</code> ，返回该字符在字符串中的位置          |
| <code>substr</code>     | 在 <code>str</code> 的 <code>pos</code> 位置开始，截取 <code>n</code> 个字符，然后将其返回 |
| <code>erase</code>      | 删除字符串的一部分，减少其长度                                                         |

```
1 // 4. 修改操作
2 // 4.1. 插入(拼接)方式: push_back append operator+=
3 // 4.2. 正向和反向查找: find() + rfind()
4 // 4.3. 截取子串: substr()
5 // 4.4. 删除: erase
6
7 void TestS4() {
8     string s0("HELLO C++");
9     cout << "origin: " << s0 << endl;
10    s0.push_back('!'); // 在s0字符串结尾尾插'!'
11 }
```

内容来源: [csdn.net](https://blog.csdn.net/ting1fengyu1)  
作者昵称: Vect  
原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/147604807>  
作者主页: <https://blog.csdn.net/ting1fengyu1>

```

12 cout << "push_back: " << s0 << endl;
13 s0.append("I like C++!"); // 结尾追加字符串"I like C++!"
14 cout << "append: " << s0 << endl;
15 s0 += "\nI like coding"; // 结尾添加字符串"\nI like coding"
16 cout << "operator+=: " << s0 << endl;
17
18 cout << s0.c_str() << endl; // 以C语言的方式打印字符串
19
20 // 获取string.cpp的后缀
21 string str("string.cpp");
22 size_t pos = str.find('.'); // 找.在str中的位置
23 string str_back(str.substr(pos, str.size() - pos)); // 找[pos,str.size()]这个区间的字符串并返回
24 cout << str_back << endl;
25
26 string url("https://cplusplus.com/");
27 size_t begin = url.find("://");
28 if (begin == string::npos) { // npos是一个静态size_t类型全局变量 值为-1 可以理解为整个字符串的结尾，超级超级大
29     cout << "无效域名" << endl;
30     return;
31 }
32
33 begin += 3;
34 string url_find(url.substr(begin, url.size() - begin - 1)); // -1去掉结尾 '/'
35 cout << url_find << endl;
36
37 // 删除url的协议前缀
38 pos = url.find("://");
39 url.erase(0, pos + 3);
40 cout << url << endl;
41
42 // 反向查找
43 string s2("nihao");
44 size_t start = s2.rfind('o');
45 size_t last = s2.rfind('n');
46 cout << s2.substr(last, start + 1) << endl; // [last,start + 1)
47
48 }

```

内容来源: csdn.net

作者昵称: Vect

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/147604807>

作者主页: <https://blog.csdn.net/ting1fengyu1>



AI写代码

输出结果：

```
origin: HELLO C++
push_back: HELLO C++!
append: HELLO C++!I like C++!
operator+=: HELLO C++!I like C++!
I like coding
HELLO C++!I like C++!
I like coding
.cpp
cplusplus.com
cplusplus.com/
nihao
```

D:\CODE\CPP\string\x64\Debug\string.exe (进程 28236)已退出，代码为 0 (0x0)。  
按任意键关闭此窗口 . . .|

CSDN @Vect.

内容来源: csdn.net

作者昵称: Vect.

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/147604807>

作者主页: <https://blog.csdn.net/ting1fengyu1>