

冠军的试炼

悟已往之不谏，知来者之可追

博客园

首页

新随笔

联系

订阅

管理

随笔 - 67 文章 - 0 评论 - 489

C++ STL快速入门

在数月之前的机试中第一次体验到STL的威力，因为自己本来一直在用C语言做开发，很多数据结构都是自己造的，比如链表、队列等，第一次接触C++ STL后发现这些数据结构都已经给我提供好了，我直接拿去调用就好了，真是超级方便。最近的项目中也遇到了STL一些容器，所以现在自己好好总结一下STL中一些最常用的容器的使用方法，方便自己日后查阅。

C++ STL中最基本以及最常用的类或容器无非就是以下几个：

- string
- vector
- set
- list
- map

下面就依次介绍它们，并给出一些最常见的最实用的使用方法，做到快速入门。

string

公告

昵称：Madcola

园龄：1年9个月

粉丝：652

关注：26

+加关注

2018年10月						
日	一	二	三	四	五	六
30	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	7	8	9	10

搜索

首先看看我们C语言一般怎么使用字符串的

```
char* s1 = "Hello SYSU!"; //创建指针指向字符串常量，这段字符串我们是不能修改的

//想要创建 可以修改的字符串，我们可以使用数组分配空间
char s2[20] = "Hello SYSU!";
//或者这样
char s3[] = "Hello SYSU!";

//当然我们也可以动态分配内存
char* s4 = (char*)malloc(20);
gets(s4);
```

C++ 标准库中的string表示可变长的字符串，它在头文件string里面。

```
#include <string>
```

用string初始化字符串分两类：用“=”号就是拷贝初始化，否则就是直接初始化。

表 3.1：初始化 string 对象的方式	
string s1	默认初始化，s1 是一个空串
string s2(s1)	s2 是 s1 的副本
string s2 = s1	等价于 s2(s1)，s2 是 s1 的副本
string s3("value")	s3 是字面值"value"的副本，除了字面值最后的那个空字符外
string s3 = "value"	等价于 s3("value")，s3 是字面值"value"的副本
string s4(n, 'c')	把 s4 初始化为由连续 n 个字符 c 组成的串

```
string s1; //初始化字符串，空字符串
string s2 = s1; //拷贝初始化，深拷贝字符串
string s3 = "I am Yasuo"; //直接初始化，s3存了字符串
string s4(10, 'a'); //s4存的字符串是aaaaaaaaaa
string s5(s4); //拷贝初始化，深拷贝字符串
string s6("I am Ali"); //直接初始化
string s7 = string(6, 'c'); //拷贝初始化，cccccc
```

常用链接

- 我的随笔
- 我的评论
- 我的参与
- 最新评论
- 我的标签

随笔分类(67)

- C++(1)
- CUDA(1)
- Linux编程(12)
- OCR系列(5)
- opencv探索(28)
- STL(2)
- 波折岁月(5)
- 工具技巧(1)
- 机器学习之旅(5)
- 深度学习(4)
- 数字图像处理(3)

随笔档案(67)

- 2018年10月 (1)
- 2018年9月 (3)
- 2018年5月 (1)
- 2018年4月 (2)
- 2018年2月 (6)

表 3.2: string 的操作

<code>os<<s</code>	将 s 写到输出流 os 当中, 返回 os
<code>is>>s</code>	从 is 中读取字符串赋给 s, 字符串以空白分隔, 返回 is
<code>getline(is, s)</code>	从 is 中读取一行赋给 s, 返回 is
<code>s.empty()</code>	s 为空返回 true, 否则返回 false
<code>s.size()</code>	返回 s 中字符的个数
<code>s[n]</code>	返回 s 中第 n 个字符的引用, 位置 n 从 0 计起
<code>s1+s2</code>	返回 s1 和 s2 连接后的结果
<code>s1=s2</code>	用 s2 的副本代替 s1 中原来的字符
<code>s1==s2</code>	如果 s1 和 s2 中所含的字符完全一样, 则它们相等; string 对象的相等性判断对字母的大小写敏感
<code>s1!=s2</code>	
<code><, <=, >, >=</code>	利用字符在字典中的顺序进行比较, 且对字母的大小写敏感

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string s1; //初始化字符串, 空字符串
    string s2 = s1; //拷贝初始化, 深拷贝字符串
    string s3 = "I am Yasuo"; //直接初始化, s3存了字符串
    string s4(10, 'a'); //s4存的字符串是aaaaaaaaaa
    string s5(s4); //拷贝初始化, 深拷贝字符串
    string s6("I am Ali"); //直接初始化
    string s7 = string(6, 'c'); //拷贝初始化, ccccccc

    //string的各种操作
    string s8 = s3 + s6; //将两个字符串合并成一个
    s3 = s6; //用一个字符串来替代另一个字符串的对用元素

    cin >> s1;
```

2018年1月 (3)
 2017年12月 (4)
 2017年11月 (3)
 2017年10月 (1)
 2017年9月 (4)
 2017年8月 (3)
 2017年7月 (5)
 2017年6月 (4)
 2017年5月 (17)
 2017年4月 (2)
 2017年2月 (2)
 2017年1月 (6)

积分与排名

积分 - 171465

排名 - 1796

最新评论

1. Re:CNN网络架构演进：从LeNet到

DenseNet

有感而发, 点赞点赞

--琅琊阁主

2. Re:【Keras】基于SegNet和U-Net的遥感图像语义分割

@贝大人我也出现这样的问题, 如果自己训练的样本, label = 1的话, 好像就没有问题...

--ajj12345

3. Re:【Keras】基于SegNet和U-Net的遥感图像语义分割

您好, 我在运行您的seget_predict.py时

(仅修改路径) 显示了很多invalid size!然后

```

cout << s1 << endl;
cout << s2 << endl;
cout << s3 << endl;
cout << s4 << endl;
cout << s5 << endl;
cout << s6 << endl;
cout << s7 << endl;
cout << s8 << endl;
cout << "s7 size = " << s7.size() << endl; //字符串长度, 不包括结束符
cout << (s2.empty() ? "This string is empty" : "This string is not empty") << endl;;

system("pause");
return 0;
}

```



string的IO操作

使用cin读入字符串时，遇到空白就停止读取。比如程序输入的是

```
"    Hello    World"
```

那么我们得到的字符串将是"Hello"，前面的空白没了，后面的world也读不出来。

如果我们想把整个hello world读进来怎么办？那就这样做

得到的pre图像是零值图像，请问是哪里出了问题呢？

--httcwr

4. Re:【OCR技术系列之二】文字定位与切割

@Madcola嗯嗯，主要是想提升对左右结构的汉字的分割准确率。英文的字母一般宽度是相同的，单词之间有空格，这可能是英文比较好切割的原因？我想问问楼主，如果有中、英、数字结合或者字体有大有小，是否投影.....

--zyh的打怪历程

5. Re:【OCR技术系列之二】文字定位与切割

@zyh的打怪历程你这个想法对于全篇都是汉字的情形会比较合适，但是一旦有中英结合以及数字的情形，效果就不好了...

--Madcola

阅读排行榜

1. 基于深度学习的目标检测技术演进：R-CNN、Fast R-CNN、Faster R-CNN(110387)
2. 卷积神经网络CNN总结(72831)
3. OpenCV探索之路（二十四）图像拼接和图像融合技术(41830)
4. OpenCV探索之路（二十三）：特征检测和特征匹配方法汇总(23313)
5. C++ STL快速入门(18472)
6. Linux编程之UDP SOCKET全攻略(18257)
7. OpenCV探索之路（十四）：绘制点、直线、几何图形(17500)

```
cin>>s1>>s2;
```

hello存在s1里，world存在s2里了。

有时我们想把一个句子存下来，又不想像上面那样创建多个string来存储单词，怎么办？

那就是用getline来获取一整行内容。

```
string str;
getline(cin, str);
cout << str << endl;
```

当把string对象和字符面值及字符串面值混在一条语句中使用，时必须确保+的两侧的运算对象至少有一个是string

```
string s1 = s2 + ", "; //正确
string s3 = "s " + ", "; //错误
string s4 = "hello" + ", " + s1; //错误
string s5 = s1 + "hello " + ", "; //改一下顺序，s1放前头，正确了，注意理解=号右边的运算顺序
```

处理string中的字符

访问字符串的每个字符

```
for (int i = 0; i < s3.size(); i++)
{
    cout << s3[i] << endl;
    s3[i] = 's';
}
```

在C语言中我都是用下标或者指针来访问数组元素，而在C++里，有个新奇的东西叫做迭代器iterator，我们可以使用它来访问容器元素。

8. OpenCV探索之路（六）：边缘检测

（canny、sobel、laplacian）(16161)

9. OpenCV探索之路（十六）：图像矫正技术深入探讨(15273)

10. OpenCV探索之路（十三）：详解掩膜

mask(14577)

评论排行榜

1. 两年波折路（考研、工作、考研）(97)

2. OpenCV探索之路（二十四）图像拼接和图像融合技术(58)

3. 【OCR技术系列之四】基于深度学习的文字识别（3755个汉字）(49)

4. 【Keras】基于SegNet和U-Net的遥感图像语义分割(37)

5. OpenCV探索之路（二十二）：制作一个类“全能扫描王”的简易扫描软件(19)

推荐排行榜

1. 两年波折路（考研、工作、考研）(88)

2. 基于深度学习的目标检测技术演进：R-CNN、Fast R-CNN、Faster R-CNN(50)

3. 卷积神经网络CNN总结(24)

4. 【Keras】基于SegNet和U-Net的遥感图像语义分割(15)

5. 【OCR技术系列之四】基于深度学习的文字识别（3755个汉字）(12)

6. 读研以来的一些感想：名校好在哪里？(12)

7. OpenCV探索之路（二十七）：皮肤检测技术(10)

表 3.6: 标准容器迭代器的运算符

<code>*iter</code>	返回迭代器 <code>iter</code> 所指元素的引用
<code>iter->mem</code>	解引用 <code>iter</code> 并获取该元素的名为 <code>mem</code> 的成员, 等价于 <code>(*iter).mem</code>
<code>++iter</code>	令 <code>iter</code> 指示容器中的下一个元素
<code>--iter</code>	令 <code>iter</code> 指示容器中的上一个元素
<code>iter1 == iter2</code>	判断两个迭代器是否相等 (不相等), 如果两个迭代器指示的是同一个元素或者它们是同一个容器的尾后迭代器, 则相等; 反之, 不相等
<code>iter1 != iter2</code>	

```
string str("hi sysu");
for (string::iterator it = str.begin(); it != str.end(); it++)
{
    cout << *it << endl;
}
```

我们也可以使用 `const_iterator` 使得访问元素时是能读不能写, 这跟常量指针意思差不多。

```
string str2("hi sysu");
for (string::const_iterator it = str2.begin(); it != str2.end(); it++)
{
    cout << *it << endl;
    *it = 'l'; //这是错误的, 不能写
}
```

8. OpenCV探索之路 (二十四) 图像拼接和图像融合技术(10)
9. 我在北京实习的四个月(8)
10. 【OCR技术系列之三】大批量生成文字训练集(7)

表 3.3: ctype 头文件中的函数

isalnum(c)	当 c 是字母或数字时为真
isalpha(c)	当 c 是字母时为真
iscntrl(c)	当 c 是控制字符时为真
isdigit(c)	当 c 是数字时为真
isgraph(c)	当 c 不是空格但可打印时为真
islower(c)	当 c 是小写字母时为真
isprint(c)	当 c 是可打印字符时为真（即 c 是空格或 c 具有可视形式）
ispunct(c)	当 c 是标点符号时为真（即 c 不是控制字符、数字、字母、可打印空白中的一种）
isspace(c)	当 c 是空白时为真（即 c 是空格、横向制表符、纵向制表符、回车符、换行符、进纸符中的一种）
isupper(c)	当 c 是大写字母时为真
isxdigit(c)	当 c 是十六进制数字时为真
tolower(c)	如果 c 是大写字母，输出对应的小写字母；否则原样输出 c
toupper(c)	如果 c 是小写字母，输出对应的大写字母；否则原样输出 c

string 还有一些很好用的函数，比如找子串

```
string sq("heoolo sdaa ss");
cout << s.find("aa", 0) << endl; //返回的是子串位置。第二个参数是查找的起始位置，如果找不到，就返回
string::npos
if (s.find("aa1", 0) == string::npos)
{
    cout << "找不到孩子串!" << endl;
}
```

vector

C++ STL中的vector好比是C语言中的数组，但是vector又具有数组没有的一些高级功能。与数组相比，vector就是一个可以不用再初始化就必须制定大小的边长数组，当然了，它还有许多高级功能。

要想用vector首先得包含头文件vector。

```
#include <vector>
```

怎么初始化？

表 3.4: 初始化 vector 对象的方法

<code>vector<T> v1</code>	<code>v1</code> 是一个空 <code>vector</code> ，它潜在的元素是 <code>T</code> 类型的，执行默认初始化
<code>vector<T> v2(v1)</code>	<code>v2</code> 中包含有 <code>v1</code> 所有元素的副本
<code>vector<T> v2 = v1</code>	等价于 <code>v2(v1)</code> ， <code>v2</code> 中包含有 <code>v1</code> 所有元素的副本
<code>vector<T> v3(n, val)</code>	<code>v3</code> 包含了 <code>n</code> 个重复的元素，每个元素的值都是 <code>val</code>
<code>vector<T> v4(n)</code>	<code>v4</code> 包含了 <code>n</code> 个重复地执行了值初始化的对象
<code>vector<T> v5{a,b,c...}</code>	<code>v5</code> 包含了初始值个数的元素，每个元素被赋予相应的初始值
<code>vector<T> v5={a,b,c...}</code>	等价于 <code>v5{a,b,c...}</code>

如果vector的元素类型是int，默认初始化为0；如果vector元素类型为string，则默认初始化为空字符串。

```
vector<int> v1;
vector<father> v2;
vector<string> v3;
vector<vector<int>> >; //注意空格。这里相当于二维数组int a[n][n];
vector<int> v5 = { 1,2,3,4,5 }; //列表初始化,注意使用的是花括号
vector<string> v6 = { "hi","my","name","is","lee" };
vector<int> v7(5, -1); //初始化为-1,-1,-1,-1,-1。第一个参数是数目，第二个参数是要初始化的值
vector<string> v8(3, "hi");
vector<int> v9(10); //默认初始化为0
vector<int> v10(4); //默认初始化为空字符串
```

如何向vector添加元素？

请使用push_back加入元素，并且这个元素是被加在数组尾部的。

```
for (int i = 0; i < 20; i++)
{
    v1.push_back(i);
}
```

vector其他的操作

表 3.5: vector 支持的操作

<code>v.empty()</code>	如果 <code>v</code> 不含有任何元素，返回真；否则返回假
<code>v.size()</code>	返回 <code>v</code> 中元素的个数
<code>v.push_back(t)</code>	向 <code>v</code> 的尾端添加一个值为 <code>t</code> 的元素
<code>v[n]</code>	返回 <code>v</code> 中第 <code>n</code> 个位置上元素的引用
<code>v1 = v2</code>	用 <code>v2</code> 中元素的拷贝替换 <code>v1</code> 中的元素
<code>v1 = {a,b,c...}</code>	用列表中元素的拷贝替换 <code>v1</code> 中的元素
<code>v1 == v2</code>	<code>v1</code> 和 <code>v2</code> 相等当且仅当它们的元素数量相同且对应位置的元素值都相同
<code>v1 != v2</code>	
<code><, <=, >, >=</code>	顾名思义，以字典顺序进行比较

访问和操作vector中的每个元素

```
for (int i = 0; i < v1.size(); i++)
{
    cout << v1[i] << endl;
    v1[i] = 100;
    cout << v1[i] << endl;
}
```

注意：只能对已存在的元素进行赋值或者修改操作，如果是要加入新元素，务必使用`push_back`。`push_back`的作用有两个：告诉编译器为新元素开辟空间、将新元素存入新空间里。

比如下面的代码是错误的，但是编译器不会报错，就像是数组越界。

```
vector<int> vec;
vec[0] = 1; //错误！
```

当然我们也可以选择使用迭代器来访问元素

```
vector<string> v6 = { "hi", "my", "name", "is", "lee" };
for (vector<string>::iterator iter = v6.begin(); iter != v6.end(); iter++)
{
    cout << *iter << endl;
    //下面两种方法都行
    cout << (*iter).empty() << endl;
}
```

```
    cout << iter->empty() << endl;
}
```

上面是正向迭代，如果我们想从后往前迭代该如何操作？

使用反向迭代器

```
for (vector<string>::reverse_iterator iter = v6.rbegin(); iter != v6.rend(); iter++)
{
    cout << *iter << endl;
}
```

vector最常用的增删操作

表 20-3 添加或删除元素的操作

操 作	说 明
push_back()	把传送为参数的对象添加到 vector 的最后
pop_back()	删除 vector 尾部的对象
erase()	删除一个或多个元素
clear()	删除所有的元素
insert()	插入一个或多个对象。这个功能是通过两个成员函数和一个函数模板提供的。函数模板可以把另一个容器的元素插入 vector。为了插入一个元素，第一个参数必须是一个指定插入位置的迭代器，第二个参数必须是要插入的对象。如前所述，在使用 vector 容器时，最好避免这个操作

```
#include <iostream>
#include <vector>
#include <string>

using namespace std;
```

```

template <typename T>
void showvector(vector<T> v)
{
    for (vector<T>::iterator it = v.begin(); it != v.end(); it++)
    {
        cout << *it;
    }
    cout << endl;
}

int main()
{
    vector<string> v6 = { "hi", "my", "name", "is", "lee" };
    v6.resize(3); //重新调整vector容量大小
    showvector(v6);

    vector<int> v5 = { 1,2,3,4,5 }; //列表初始化,注意使用的是花括号
    cout << v5.front() << endl; //访问第一个元素
    cout << v5.back() << endl; //访问最后一个元素

    showvector(v5);
    v5.pop_back(); //删除最后一个元素
    showvector(v5);
    v5.push_back(6); //加入一个元素并把它放在最后
    showvector(v5);
    v5.insert(v5.begin()+1,9); //在第二个位置插入新元素
    showvector(v5);
    v5.erase(v5.begin() + 3); //删除第四个元素
    showvector(v5);
    v5.insert(v5.begin() + 1, 7,8); //连续插入7个8
    showvector(v5);
    v5.clear(); //清除所有内容
    showvector(v5);

    system("pause");
    return 0;
}

```

```
C:\Windows\system32\cmd.exe
```

```
himyname  
1  
5  
12345  
1234  
12346  
192346  
19246  
188888889246
```

```
请按任意键继续. . .
```

注意：虽然vector对象可以动态增长，但是也或有一点副作用：已知的一个限制就是不能再范围for循环中向vector对象添加元素。另外一个限制就是任何一种可能改变vector对象容量的操作，不如push_back，都会使该迭代器失效。

总而言之就是：但凡使用了迭代器的循环体，都不要向迭代器所属的容器添加元素！

C++中push_back和insert两个有什么区别？

顾名思义push_back把元素插入容器末尾，insert把元素插入任何你指定的位置。

不过push_back速度一般比insert快。如果能用push_back尽量先用push_back。

set

set跟vector差不多，它跟vector的唯一区别就是，set里面的元素是有序的且唯一的，只要你往set里添加元素，它就会自动排序，而且，如果你添加的元素set里面本来就存在，那么这次添加操作就不执行。要想用set先加个头文件set。

```
#include <set>
```

```
#include <iostream>
```

```
#include <set>
#include <string>

using namespace std;

template <typename T>
void showset(set<T> v)
{
    for (set<T>::iterator it = v.begin(); it != v.end(); it++)
    {
        cout << *it;
    }
    cout << endl;
}

int main()
{
    set<int> s1{9,8,1,2,3,4,5,5,5,6,7,7 }; //自动排序,从小到大,剔除相同项
    showset(s1);
    set<string> s2{ "hello","sysy","school","hello" }; //字典序排序
    showset(s2);
    s1.insert(9); //有这个值了,do nothing
    showset(s1);
    s2.insert("aaa"); //没有这个字符串,添加并且排序
    showset(s2);

    system("pause");
    return 0;
}
```

cmd C:\Windows\system32\cmd.exe

```
123456789
hello school sysy
123456789
aaa hello school sysy
请按任意键继续. . .
```

list

list就是链表，在C语言中我们想使用链表都是自己去实现的，实现起来倒不难，但是如果有现成的高效的链表可以使用的话，我们就不需要重复造轮子了。STL就提供了list容器给我们。

list是一个双向链表，而单链表对应的容器则是forward_list。

list即双向链表的优点是插入和删除元素都比较快捷，缺点是不能随机访问元素。

初始化方式就大同小异了，跟vector基本一样。要想用list先加个头文件list。

```
#include <list>
```

```
#include <iostream>
#include <list>
#include <string>

using namespace std;

template <typename T>
void showlist(list<T> v)
{
    for (list<T>::iterator it = v.begin(); it != v.end(); it++)
```

```

    {
        cout << *it;
    }
    cout << endl;
}

int main()
{
    list<int> l1{ 1,2,3,4,5,5,6,7,7 };
    showlist(l1);
    list<double> l2;
    list<char> l3(10);
    list<int> l4(5, 10); //将元素都初始化为10
    showlist(l4);

    system("pause");
    return 0;
}

```

CA C:\Windows\system32\cmd.exe

```

123455677
1010101010
请按任意键继续. . .

```

值得注意的是，list容器不能调用algorithm下的sort函数进行排序，因为sort函数要求容器必须可以随机存储，而list做不到。所以，list自己做了一个自己用的排序函数，用法如下：

```

list<int> l1{ 8,5,7,6,1,2,3,4,5,5,6,7,7 };
l1.sort();

```

map

map运用了哈希表地址映射的思想，也就是key-value的思想，来实现的。

首先给出map最好用也最最常用的用法例子，就是用字符串作为key去查询操作对应的value。

要使用map得先加个头文件map。

```
#include <map>
```

```
#include <iostream>
#include <map>
#include <string>

using namespace std;

void showmap(map<string, int> v)
{
    for (map<string, int>::iterator it = v.begin(); it != v.end(); it++)
    {
        cout << it->first << " " << it->second << endl; //注意用法，不是用*it来访问了。first表示的是key，second存的是value
    }
    cout << endl;
}

int main()
{
    map<string, int> m1; //<>里的第一个参数表示key的类型，第二个参数表示value的类型
    m1["Kobe"] = 100;
    m1["James"] = 99;
    m1["Curry"] = 98;

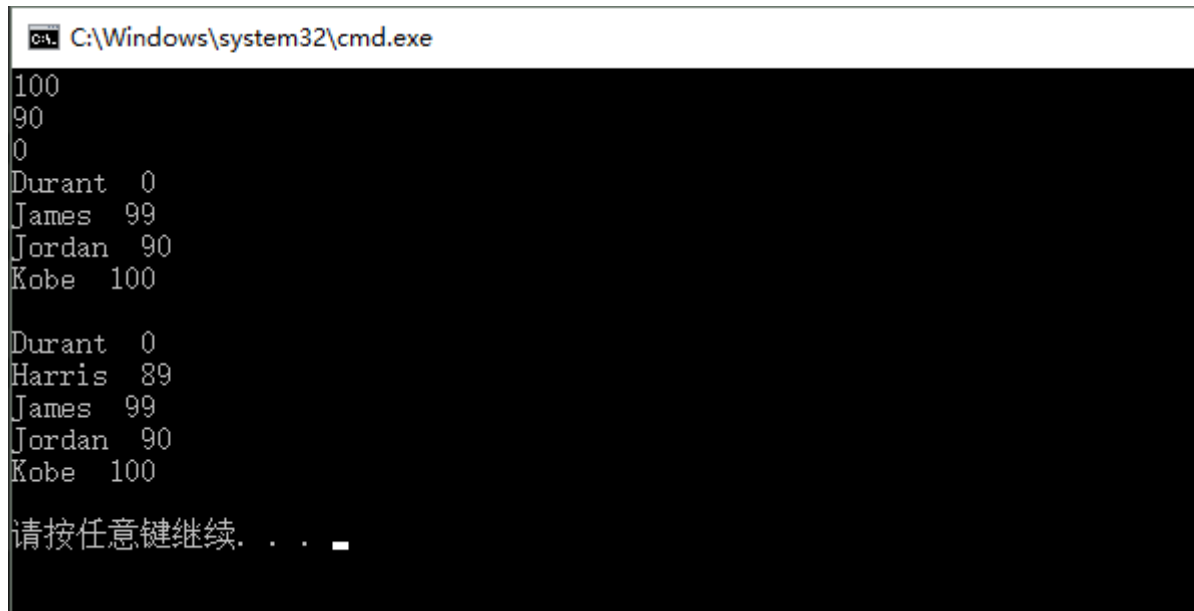
    string s("Jordan");
    m1[s] = 90;

    cout << m1["Kobe"] << endl;
    cout << m1["Jordan"] << endl;
    cout << m1["Durant"] << endl; //不存在这个key，就显示0

    m1.erase("Curry"); //通过关键字来删除
```



```
showmap(m1);  
m1.insert(pair<string, int>("Harris", 89)); //也可以通过insert函数来实现增加元素  
showmap(m1);  
m1.clear(); //清空全部  
  
system("pause");  
return 0;  
}
```



```
C:\Windows\system32\cmd.exe  
100  
90  
0  
Durant 0  
James 99  
Jordan 90  
Kobe 100  
  
Durant 0  
Harris 89  
James 99  
Jordan 90  
Kobe 100  
  
请按任意键继续. . .
```

如果想看看某个存不存在某个key，可以用count来判断

```
if (m1.count("Lee"))  
{  
    cout << "Lee is in m1!" << endl;  
}  
else  
{  
    cout << "Lee do not exist!" << endl;  
}
```

用迭代器来访问元素

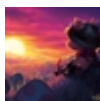
```
for (map<string, int>::iterator it = m1.begin(); it != m1.end(); it++)
{
    cout << it->first<<" " << it->second << endl; //注意用法，不是用*it来访问了。first表示的是key，
    second存的是value
}
```

分类: STL

好文要顶

关注我

收藏该文



Madcola

关注 - 26

粉丝 - 652

[+加关注](#)

« 上一篇: [OpenCV探索之路 \(十六\): 图像矫正技术深入探讨](#)

» 下一篇: [Linux编程之有限状态机FSM的理解与实现](#)

posted @ 2017-06-02 17:10 Madcola 阅读(18475) 评论(10) 编辑 收藏

4

0

评论列表

#1楼 2018-03-19 15:44 **Y先森0.0**

谢谢，写的不错

支持(0) 反对(0)

#2楼 2018-04-10 13:52 **huang_lei**

谢谢，很容易理解了

支持(0) 反对(0)

#3楼 2018-05-09 06:38 **ZincSabian**

感谢，作为一个刚从pascal-->C++的菜鸡，stl一直是个大锅pwp

支持(0) 反对(0)

#4楼 2018-07-20 13:06 **黄zzzz**

学习的时候没认真学STL，今年毕业了干了几个月前端觉得工作不是很喜欢，又想重新捡起当初学的C++换个工作，SYSU学子共勉吧

支持(1) 反对(0)

#5楼 2018-07-24 14:13 **云中之君**

@ 黄zzzz
与君共勉

支持(0) 反对(0)

#6楼 2018-08-01 09:56 **gvanni**

你好 这篇博客中的图片是来自哪本书咩？之前看过电子版 想不起来书名了

支持(0) 反对(0)

#7楼 2018-08-01 10:03 **gvanni**

急求急求啊~~~麻烦回复一个咯~~~

支持(0) 反对(0)

#8楼 2018-08-10 00:49 **大母鸡翅膀**

@ gvanni
好像是C++标准库

支持(0) 反对(0)

#9楼 2018-09-24 21:54 **eddiechiu**

```
set<int> s1{9,8,1,2,3,4,5,5,5,6,7,7};
```

这种初始化方法会报错，难道是版本不同了么？

只能用.insert()

支持(0) 反对(0)

#10楼[[楼主](#)] 2018-09-25 08:32 **Madcola**

@ eddiechiu

C++11支持这样的初始化方式

支持(0) 反对(0)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问网站首页](#)。

【推荐】超50万VC++源码: 大型组态工控、电力仿真CAD与GIS源码库！

【推荐】华为云11.11普惠季 血拼风暴 一促即发

【拼团】腾讯云服务器拼团活动又双叒叕来了！

【推荐】腾讯云新注册用户域名抢购1元起



腾讯云AMD云服务器

节省IT成本30%
1核1G AMD机型**0.57元/天**起

立即抢购

最新IT新闻:

- 金融服务创企罗宾汉收入来源引争议：或造成利益冲突
 - 长春长生被罚91亿 吊销《药品生产许可证》
 - 曾获迪士尼投资的VR创企Jaunt XR将裁员：未来专注AR
 - 网易云音乐获新一轮融资 投资方表示长期看好
 - 科技股收盘|美三大股指涨幅创半年多新高 科技股集体大涨
- » 更多新闻...



最新知识库文章:

- 为什么说 Java 程序员必须掌握 Spring Boot ?
 - 在学习中, 有一个比掌握知识更重要的能力
 - 如何招到一个靠谱的程序员
 - 一个故事看懂“区块链”
 - 被踢出去的用户
- » 更多知识库文章...

