* 英语冲 ！
* 加油！当自己的实力不能满足自己的目标时，就**静下心**去学习！
* 倒计时 500 天



@[toc]

# string容器

## string基本概念

**本质：**

* string是C++风格的字符串，而string本质上是一个类

string和char \* 区别：

* char \* 是一个指针
* string是一个类，类内部封装了char\*，管理这个字符串，是一个char\*型的容器。

**特点：**

* string 类内部封装了很多成员方法
* 例如：查找find，拷贝copy，删除delete 替换replace，插入insert
* string管理char\*所分配的内存，不用担心复制越界和取值越界等，由类内部进行负责

## string构造函数

构造函数原型：

* string(); //创建一个空的字符串 例如: string str;  
  string(const char\* s); //使用字符串s初始化
* string(const string& str); //使用一个string对象初始化另一个string对象
* string(int n, char c); //使用n个字符c初始化

**示例：**

#include <string>  
//string构造  
void test01()  
{  
 string s1; //创建空字符串，调用无参构造函数  
 cout << "str1 = " << s1 << endl;  
  
 const char\* str = "hello world";  
 string s2(str); //把c\_string转换成了string  
  
 cout << "str2 = " << s2 << endl;  
  
 string s3(s2); //调用拷贝构造函数  
 cout << "str3 = " << s3 << endl;  
  
 string s4(10, 'a');  
 cout << "str3 = " << s3 << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：string的多种构造方式没有可比性，灵活使用即可

## string赋值操作

功能描述：

* 给string字符串进行赋值

赋值的函数原型：

* string& operator=(const char\* s); //char\*类型字符串 赋值给当前的字符串
* string& operator=(const string &s); //把字符串s赋给当前的字符串
* string& operator=(char c); //字符赋值给当前的字符串
* string& assign(const char \*s); //把字符串s赋给当前的字符串
* string& assign(const char \*s, int n); //把字符串s的前n个字符赋给当前的字符串
* string& assign(const string &s); //把字符串s赋给当前字符串
* string& assign(int n, char c); //用n个字符c赋给当前字符串

**示例：**

//赋值  
void test01()  
{  
 string str1;  
 str1 = "hello world";  
 cout << "str1 = " << str1 << endl;  
  
 string str2;  
 str2 = str1;  
 cout << "str2 = " << str2 << endl;  
  
 string str3;  
 str3 = 'a';  
 cout << "str3 = " << str3 << endl;  
  
 string str4;  
 str4.assign("hello c++");  
 cout << "str4 = " << str4 << endl;  
  
 string str5;  
 str5.assign("hello c++",5);  
 cout << "str5 = " << str5 << endl;  
  
  
 string str6;  
 str6.assign(str5);  
 cout << "str6 = " << str6 << endl;  
  
 string str7;  
 str7.assign(5, 'x');  
 cout << "str7 = " << str7 << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：string的赋值方式很多，operator= 为常用

## string查找和替换

**功能描述：**

* 查找：查找指定字符串是否存在
* 替换：在指定的位置替换字符串

**函数原型：**

* int find(const string& str, int pos = 0) const;   
  查找str第一次出现位置,从pos开始查找
* int find(const char\* s, int pos = 0) const;   
  查找s第一次出现位置,从pos开始查找
* int find(const char\* s, int pos, int n) const;   
  从pos位置查找s的前n个字符第一次位置
* int find(const char c, int pos = 0) const;   
  查找字符c第一次出现位置
* int rfind(const string& str, int pos = npos) const;   
  查找str最后一次位置,从pos开始查找
* int rfind(const char\* s, int pos = npos) const;   
  查找s最后一次出现位置,从pos开始查找
* int rfind(const char\* s, int pos, int n) const;   
  从pos查找s的前n个字符最后一次位置
* int rfind(const char c, int pos = 0) const;   
  查找字符c最后一次出现位置
* string& replace(int pos, int n, const string& str);   
  替换从pos开始n个字符为字符串str
* string& replace(int pos, int n,const char\* s);   
  替换从pos开始的n个字符为字符串s

**示例：**

//查找和替换  
void test01()  
{  
 //查找  
 string str1 = "abcdefgde";  
  
 int pos = str1.find("de");  
  
 if (pos == -1)  
 {  
 cout << "未找到" << endl;  
 }  
 else  
 {  
 cout << "pos = " << pos << endl;  
 }  
   
  
 pos = str1.rfind("de");  
  
 cout << "pos = " << pos << endl;  
  
}  
  
void test02()  
{  
 //替换  
 string str1 = "abcdefgde";  
 str1.replace(1, 3, "1111");  
  
 cout << "str1 = " << str1 << endl;  
}  
  
int main() {  
  
 //test01();  
 //test02();  
  
 system("pause");  
  
 return 0;  
}

**总结**：

* find查找是从左往后，rfind从右往左
* find找到字符串后返回查找的第一个字符位置，找不到返回-1
* replace在替换时，要指定从哪个位置起，多少个字符，替换成什么样的字符串

## string字符串比较

**功能描述：**

* 字符串之间的比较

**比较方式：**

* 字符串比较是按字符的ASCII码进行对比

= 返回 0

> 返回 1

< 返回 -1

**函数原型：**

* int compare(const string &s) const; //与字符串s比较
* int compare(const char \*s) const; //与字符串s比较

**示例：**

//字符串比较  
void test01()  
{  
  
 string s1 = "hello";  
 string s2 = "aello";  
  
 int ret = s1.compare(s2);  
  
 if (ret == 0) {  
 cout << "s1 等于 s2" << endl;  
 }  
 else if (ret > 0)  
 {  
 cout << "s1 大于 s2" << endl;  
 }  
 else  
 {  
 cout << "s1 小于 s2" << endl;  
 }  
  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结**：字符串对比主要是用于比较两个字符串是否相等，判断谁大谁小的意义并不是很大

## string字符存取

string中单个字符存取方式有两种

* char& operator[](int n); 通过[]方式取字符
* char& at(int n); 通过at方法获取字符

**示例：**

void test01()  
{  
 string str = "hello world";  
  
 for (int i = 0; i < str.size(); i++)  
 {  
 cout << str[i] << " ";  
 }  
 cout << endl;  
  
 for (int i = 0; i < str.size(); i++)  
 {  
 cout << str.at(i) << " ";  
 }  
 cout << endl;  
  
  
 //字符修改  
 str[0] = 'x';  
 str.at(1) = 'x';  
 cout << str << endl;  
   
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：string字符串中单个字符存取有两种方式，利用 [ ] 或 at

## string插入和删除

**功能描述：**

* 对string字符串进行插入和删除字符操作

**函数原型：**

* string& insert(int pos, const char\* s); //插入字符串
* string& insert(int pos, const string& str); //插入字符串
* string& insert(int pos, int n, char c); //在指定位置插入n个字符c
* string& erase(int pos, int n = npos); //删除从Pos开始的n个字符

**示例：**

//字符串插入和删除  
void test01()  
{  
 string str = "hello";  
 str.insert(1, "111");  
 cout << str << endl;  
  
 str.erase(1, 3); //从1号位置开始3个字符  
 cout << str << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结**: 插入和删除的起始下标都是从0开始

## string子串

**功能描述：**

* 从字符串中获取想要的子串

**函数原型：**

* string substr(int pos = 0, int n = npos) const; //返回由pos开始的n个字符组成的字符串

**示例：**

//子串  
void test01()  
{  
  
 string str = "abcdefg";  
 string subStr = str.substr(1, 3);  
 cout << "subStr = " << subStr << endl;  
  
 string email = "hello@sina.com";  
 int pos = email.find("@");  
 string username = email.substr(0, pos);  
 cout << "username: " << username << endl;  
  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结** : 灵活的运用求子串功能，可以在实际开发中获取有效的信息

# vector容器

## vector基本概念

**功能：**

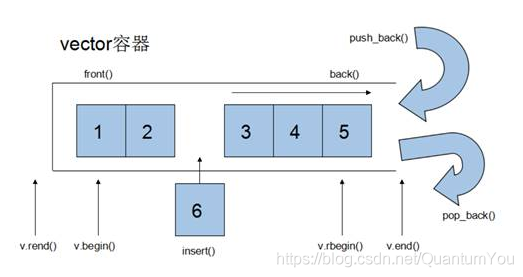
* vector数据结构和**数组非常相似**，也称为**单端数组**

**vector与普通数组区别：**

* 不同之处在于数组是静态空间，而vector可以**动态扩展**

**动态扩展：**

* 并不是在原空间之后续接新空间，而是找更大的内存空间，然后将原数据拷贝新空间，释放原空间



* vector容器的迭代器是支持随机访问的迭代器

## vector构造函数

**功能描述：**

* 创建vector容器

**函数原型：**

* vector<T> v; //采用模板实现类实现，默认构造函数
* vector(v.begin(), v.end()); //将v[begin(), end())区间中的元素拷贝给本身。
* vector(n, elem); //构造函数将n个elem拷贝给本身。
* vector(const vector &vec); //拷贝构造函数。

**示例：**

#include <vector>  
  
void printVector(vector<int>& v) {  
  
 for (vector<int>::iterator it = v.begin(); it != v.end(); it++) {  
 cout << \*it << " ";  
 }  
 cout << endl;  
}  
  
void test01()  
{  
 vector<int> v1; //无参构造  
 for (int i = 0; i < 10; i++)  
 {  
 v1.push\_back(i);  
 }  
 printVector(v1);  
  
 vector<int> v2(v1.begin(), v1.end());  
 printVector(v2);  
  
 vector<int> v3(10, 100);  
 printVector(v3);  
   
 vector<int> v4(v3);  
 printVector(v4);  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结** : vector的多种构造方式没有可比性，灵活使用即可

## vector赋值操作

**功能描述：**

* 给vector容器进行赋值

**函数原型：**

* vector& operator=(const vector &vec);//重载等号操作符
* assign(beg, end); //将[beg, end)区间中的数据拷贝赋值给本身。
* assign(n, elem); //将n个elem拷贝赋值给本身。

**示例：**

#include <vector>  
  
void printVector(vector<int>& v) {  
  
 for (vector<int>::iterator it = v.begin(); it != v.end(); it++) {  
 cout << \*it << " ";  
 }  
 cout << endl;  
}  
  
//赋值操作  
void test01()  
{  
 vector<int> v1; //无参构造  
 for (int i = 0; i < 10; i++)  
 {  
 v1.push\_back(i);  
 }  
 printVector(v1);  
  
 vector<int>v2;  
 v2 = v1;  
 printVector(v2);  
  
 vector<int>v3;  
 v3.assign(v1.begin(), v1.end());  
 printVector(v3);  
  
 vector<int>v4;  
 v4.assign(10, 100);  
 printVector(v4);  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

**总结**： vector赋值方式比较简单，使用operator=，或者assign都可以

## vector容量和大小

**功能描述：**

* 对vector容器的容量和大小操作

**函数原型：**

* empty(); //判断容器是否为空
* capacity(); //容器的容量
* size(); //返回容器中元素的个数
* resize(int num); //重新指定容器的长度为num，若容器变长，则以默认值填充新位置。
* //如果容器变短，则末尾超出容器长度的元素被删除。
* resize(int num, elem); //重新指定容器的长度为num，若容器变长，则以elem值填充新位置。
* //如果容器变短，则末尾超出容器长度的元素被删除

**示例：**

#include <vector>  
  
void printVector(vector<int>& v) {  
  
 for (vector<int>::iterator it = v.begin(); it != v.end(); it++) {  
 cout << \*it << " ";  
 }  
 cout << endl;  
}  
  
void test01()  
{  
 vector<int> v1;  
 for (int i = 0; i < 10; i++)  
 {  
 v1.push\_back(i);  
 }  
 printVector(v1);  
 if (v1.empty())  
 {  
 cout << "v1为空" << endl;  
 }  
 else  
 {  
 cout << "v1不为空" << endl;  
 cout << "v1的容量 = " << v1.capacity() << endl;  
 cout << "v1的大小 = " << v1.size() << endl;  
 }  
  
 //resize 重新指定大小 ，若指定的更大，默认用0填充新位置，可以利用重载版本替换默认填充  
 v1.resize(15,10);  
 printVector(v1);  
  
 //resize 重新指定大小 ，若指定的更小，超出部分元素被删除  
 v1.resize(5);  
 printVector(v1);  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：

* 判断是否为空 --- empty
* 返回元素个数 --- size
* 返回容器容量 --- capacity
* 重新指定大小 --- resize

## vector插入和删除

**功能描述：**

* 对vector容器进行插入、删除操作

**函数原型：**

* push\_back(ele); //尾部插入元素ele
* pop\_back(); //删除最后一个元素
* insert(const\_iterator pos, ele); //迭代器指向位置pos插入元素ele
* insert(const\_iterator pos, int count,ele);//迭代器指向位置pos插入count个元素ele
* erase(const\_iterator pos); //删除迭代器指向的元素
* erase(const\_iterator start, const\_iterator end);//删除迭代器从start到end之间的元素
* clear(); //删除容器中所有元素

**示例：**

#include <vector>  
  
void printVector(vector<int>& v) {  
  
 for (vector<int>::iterator it = v.begin(); it != v.end(); it++) {  
 cout << \*it << " ";  
 }  
 cout << endl;  
}  
  
//插入和删除  
void test01()  
{  
 vector<int> v1;  
 //尾插  
 v1.push\_back(10);  
 v1.push\_back(20);  
 v1.push\_back(30);  
 v1.push\_back(40);  
 v1.push\_back(50);  
 printVector(v1);  
 //尾删  
 v1.pop\_back();  
 printVector(v1);  
 //插入  
 v1.insert(v1.begin(), 100);  
 printVector(v1);  
  
 v1.insert(v1.begin(), 2, 1000);  
 printVector(v1);  
  
 //删除  
 v1.erase(v1.begin());  
 printVector(v1);  
  
 //清空  
 v1.erase(v1.begin(), v1.end());  
 v1.clear();  
 printVector(v1);  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：

* 尾插 --- push\_back
* 尾删 --- pop\_back
* 插入 --- insert (位置迭代器)
* 删除 --- erase （位置迭代器）
* 清空 --- clear

## vector数据存取

**功能描述：**

* 对vector中的数据的存取操作

**函数原型：**

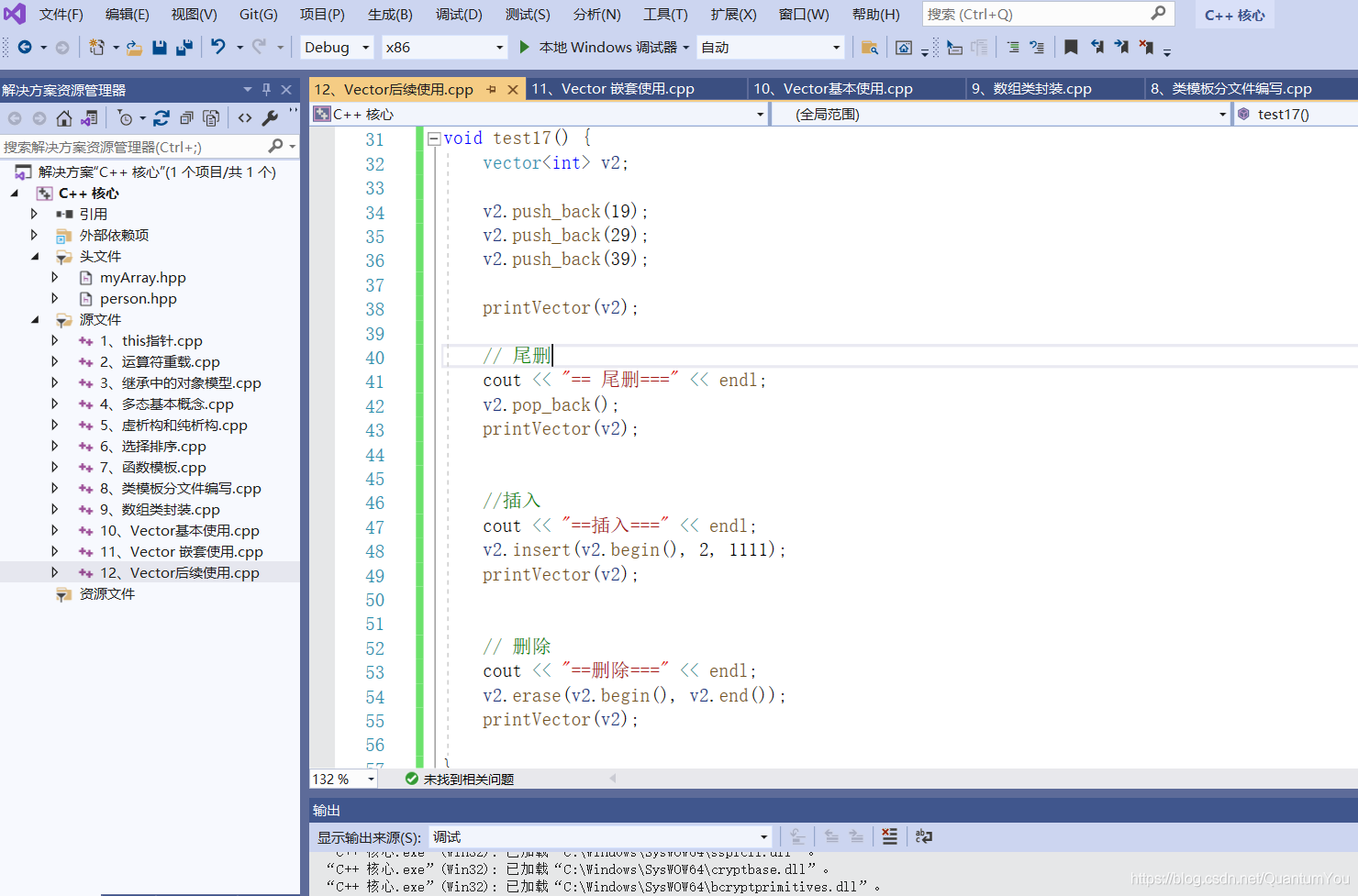
* at(int idx); //返回索引idx所指的数据
* operator[]; //返回索引idx所指的数据
* front(); //返回容器中第一个数据元素
* back(); //返回容器中最后一个数据元素

**示例：**

#include <vector>  
  
void test01()  
{  
 vector<int>v1;  
 for (int i = 0; i < 10; i++)  
 {  
 v1.push\_back(i);  
 }  
  
 for (int i = 0; i < v1.size(); i++)  
 {  
 cout << v1[i] << " ";  
 }  
 cout << endl;  
  
 for (int i = 0; i < v1.size(); i++)  
 {  
 cout << v1.at(i) << " ";  
 }  
 cout << endl;  
  
 cout << "v1的第一个元素为： " << v1.front() << endl;  
 cout << "v1的最后一个元素为： " << v1.back() << endl;  
}  
  
int main() {  
  
 test01();  
  
 system("pause");  
  
 return 0;  
}

总结：

* 除了用迭代器获取vector容器中元素，[ ]和at也可以
* front返回容器第一个元素
* back返回容器最后一个元素



## vector互换容器

**功能描述：**

* 实现两个容器内元素进行互换

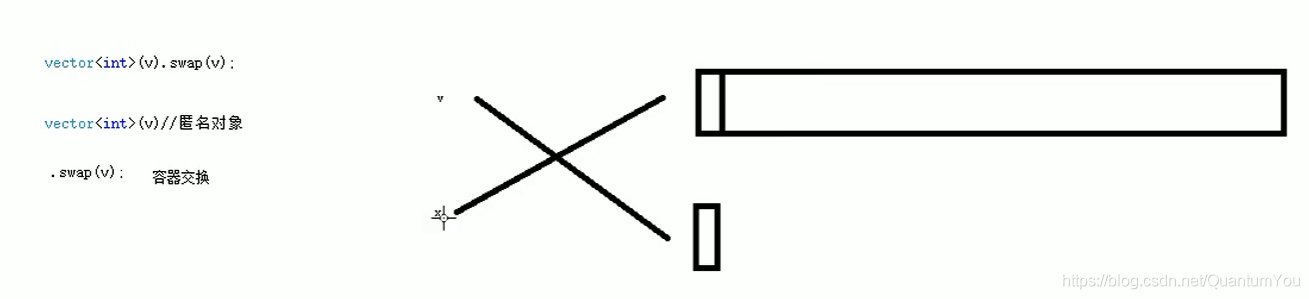
**函数原型：**

* swap(vec); // 将vec与本身的元素互换

**示例：**

#include <vector>  
  
void printVector(vector<int>& v) {  
  
 for (vector<int>::iterator it = v.begin(); it != v.end(); it++) {  
 cout << \*it << " ";  
 }  
 cout << endl;  
}  
  
void test01()  
{  
 vector<int>v1;  
 for (int i = 0; i < 10; i++)  
 {  
 v1.push\_back(i);  
 }  
 printVector(v1);  
  
 vector<int>v2;  
 for (int i = 10; i > 0; i--)  
 {  
 v2.push\_back(i);  
 }  
 printVector(v2);  
  
 //互换容器  
 cout << "互换后" << endl;  
 v1.swap(v2);  
 printVector(v1);  
 printVector(v2);  
}  
  
void test02()  
{  
 vector<int> v;  
 for (int i = 0; i < 100000; i++) {  
 v.push\_back(i);  
 }  
  
 cout << "v的容量为：" << v.capacity() << endl;  
 cout << "v的大小为：" << v.size() << endl;  
  
 v.resize(3);  
  
 cout << "v的容量为：" << v.capacity() << endl;  
 cout << "v的大小为：" << v.size() << endl;  
  
 //收缩内存  
 vector<int>(v).swap(v); //匿名对象  
  
 cout << "v的容量为：" << v.capacity() << endl;  
 cout << "v的大小为：" << v.size() << endl;  
}  
  
int main() {  
  
 test01();  
  
 test02();  
  
 system("pause");  
  
 return 0;  
}

总结：swap可以使两个容器互换，可以达到实用的收缩内存效果

空间互换的原理：利用匿名对象，程序结束后自动释放空间  


## vector预留空间

**功能描述：**

* 减少vector在动态扩展容量时的扩展次数

**函数原型：**

* reserve(int len);//容器预留len个元素长度，预留位置不初始化，元素不可访问。

**示例：**

#include <vector>  
  
void test01()  
{  
 vector<int> v;  
  
 //预留空间  
 v.reserve(100000);  
  
 int num = 0;  
 int\* p = NULL;  
 for (int i = 0; i < 100000; i++) {  
 v.push\_back(i);  
 if (p != &v[0]) {  
 p = &v[0];  
 num++;  
 }  
 }  
  
 cout << "num:" << num << endl;  
}  
  
int main() {  
  
 test01();  
   
 system("pause");  
  
 return 0;  
}

总结：如果数据量较大，可以一开始利用reserve预留空间