字符串 string 用法（与迭代器有关）

string 字符串类型一致是笔试面试的高频考点，如果要深究，其包含的知识点非常多。处理此类问题，只掌握基本的用法是远远不够的，不仅编程效率低、鲁棒性和通用性也比较差。此篇博文作为进阶篇的笔记，供自己时常复习。

基础篇

1、string[n]

```
string a[3];            相当于开了 3 个 string 类型的字符串
for(int i = 1; i <= 3; i++) {
    cin >> a[1];
}
```

2、字符字面值和字符串字面值

字符字面值：又称 **char** 型字面值，由单引号括起来，比如 **‘A’**。

字符串字面值：常量字符构成的数组。编译器会在**string** 类型字符串后面加一个空字符 **‘\0’**，因此字符串的实际长度要比它的内容多 **1**。比如 **char** 类型的 **‘A’**，和 **string** 类型的**"A"**，**'A'**就是一个单纯的字符 **A** ，而**"A"** 包含两个字符：一个是字母 **A**，一个是空字符 **‘\0’**。

比如：

```
int main() {
    string str = "abc";
    cout << str.size() << endl; // 输出3
}
```

输出为什么是 **3** 呢？不是说在字符串后面会自动加一个空格吗？不应该是 **4** 吗？其实，**size()** 输出的仅仅是 **string** 类型变量的字符个数。再次说明一下：

其实，不管是执行拷贝初始化还是直接初始化， str 都是 “abc”的副本，除了最后那个空格。也就是说，最后那个空格是不会被赋值给别的变量的

3、C 风格的字符串

C 风格的字符串在 C++11标准中，是不建议使用的，因为它不仅操作起来不方便，而且极易引发程序漏洞。但是呢？很多旧程序或者很多笔试面试题，都包含了很多 c风格的字符串。所以，学习 C 风格的字符串还是很有益的。

C 风格的字符串不是一种类型，而是为了表达和使用字符串而约定俗成的一种写法。 C 风格字符串有两种，一种是字符串常量，另一种是末尾添加了 ‘\0’的字符数组。如下：

```
char *str0 = "abc";

char str1[] = "abc";

char str2[] = {'a','b','c','\0'};
```

以上三种定义是完全一样的。但是如果这样定义：

```
har str[] = {'a','b','c'};
```

则不属于 C 风格的字符串。也意味着不能使用下面的几个库函数

C 风格字符串包含在 cstring 头文件中。主要包含以下 4 个函数：

（1）strlen (p);　　// 返回 p 的长度，空字符不包含在内

（2）strcmp(p1，p2);　　// 比较平p1 ,p2 的相等性。如果相等，返回 0 ; 如果 p1 > p2，返回一个正值，否则返回一个负值

（3）strcat (p1,p2);　　// 将p2 附加到 p1 上，返回 p1

（4）strcpy(p1,p2);　　// 将p2 拷贝给 p1 ，返回p1

具体用法不赘述。但是有一点很重要，传入此类函数中的指针，必须指向以空字符结束的数组。比如：

```
char a[] = {'a','b','c'};

cout <<strlen(a) << endl;
```

就会出现严重的错误，或者输出一个随机值，但是基本上不会是3 。因为字符数组 a 没有以空字符结尾。但是如果:

```
char a[] = {'a','b','c','\0'};
```

输出的就是 3 了。如果这样更改，也是没问题的，每一个都是正确的：

```
char a[] = { "abc" };

char a[] = "abc" ;

char *a = { "abc" };

char *a = "abc";
```

但是，如果这样写：

```
char *a = { 'a', 'b','c', '\0' };
```

显然，已经错到外星球了。

4、混用 string 类型和 C 风格字符串

（1）可以用以空字符结束的字符数组来初始化一个 string 对象。但是，不能用string对象直接初始化指向字符的指针。比如:

```
int main() {
    char *s = "abc";
    string str = s;
    cout << str << endl;  // 输出 abc
}
```

这样是可以的。但是如果反过来，比如这样：

```
int main() {
    string str = "abc";
    char *s = str;  // 编译不通过
    cout << s << endl;
}
```

是不可以的。那怎么办呢？可以这样修改：

```
int main() {
    string str = "abc";
    const char *s = str.c_str();  // 加const 是为了确保我们不会改变字符数组的内容，如果没有
const ，编译不通过。
    cout << s << endl;   // 输出 abc
}
```

其中，

（1）**c_str()** 函数的返回值是一个 **C** 风格的字符串。也就是说，该函数返回的是个指针，该指针指向以空格结尾的空字符串（或字符数组）

（2）为什么要加 **const** ，因为，**c_str()** 函数返回的指针类型是 **const char*** 类型的，为了确保我们不会改变字符数组的内容。

但是，程序如果这样更改呢？

```
int main() {
    string str = "abc";
    const char *s = str.c_str();
    str = "zxc";
    cout << s << endl;  // 输出为 zxc
}
```

输出就变了，变成了 **zxc** 。因为我们无法保证 **c_str()** 返回的数组一直有效，事实上，如果在后续改变了 **s** 的值，就可能让之前返回的数组失去效用。所以，如果执行完 **c_str()** 后想一直都用其返回的数组，最好将原来的数组重新拷贝一份。如果这样的话，就涉及到内存分配的东西了，先不讲。

5、string 类型的基本操作

（1）两个 string 相加

在**C++** 中，**string** 表示可变长的字符序列。也就是说，对于**string**类型的变量，可以在后面 **+** 一个字符或者字符串 ，但是必须确保加法运算符的两侧至少有一个是 **String** 类型，不过，字符串字面值并不是标准库类型 **string** 的对象。几个实例如下所示：

```
string s1 = "hello";

string s2 = s1 +',';  // 正确的

string s3 = "hello" + ',';  // 不正确，必须确保 + 两侧至少有一个是 string 类型

string s4 = s1 +',' + "hello";//正确，因为先执行的是 s1 +','结果是个字符串类型了。但是如果"hello" + ','+s1 显然就是错误的
```

（2）利用范围for() 处理 string 对象中的字符

处理所有的字符，可以利用范围 for 循环。比如:

```
string str = "hello";
for(auto &c : str)
    处理字符 c
```

其中，auto 表示让编译器自己来决定变量 c的类型，显然，这里的类型是 char ,每一次迭代，str 的下一个字符就会被拷贝到c中，因此该循环可以读成“对于字符串 str 中的每一个字符”。这部分很简单，遗忘的部分参考《C++ Prime 第五版》 P83.

（3）利用下标运算符

String 对象的下标是从 0 开始计数，str[size()-1] 表示最后一个字符。在此，需要注意两点：

（1）下标必须大于等于0 且小于size();

（2）访问字符之前，必须检查 s 是否为空。如果不为空，才能操作该位置上的字符。比如说string s; s[0] = 'i'，显然是错误的。比如下面这段程序：

```
string s1 = "hello";
string s2;
for(int i = 0;i < s1.size();++i)
        s2[i] = s1[i];
cout <<s2<<endl;
```

显然是错误的。因为定义的是空 s2 ，它并没有下标。那怎么实现这个功能呢？可以这样：

```
string s1 = "hello";
string s2;
for(int i = 0;i < s1.size();++i)
        s2[i] += s1[i];     // 多了一个 + 号
cout <<s2<<endl;
```

（3）保证下标的合法性，可以将下标的类型一直设置为 **string::size_type** ，因为此类型是无符号数，可以确保大于等于**0**。

（4）使用迭代器操作 **string** 的某一字符

C++ 标准库容器都支持迭代器，**string** 虽然不是标准库类型，但是同样支持迭代器。和指针类型很像，迭代器提供的是对对象的间接访问。而这个对象，就是 **string** 对象中的字符。使用迭代器可以访问某个元素，也可以从一个元素移动到另外一个元素。

支持迭代器的类型，也同时拥有迭代器的成员：**begin()** 和 **end()** ，其中，**begin()** 指向第一个元素，而**end()** 指向的是尾元素的下一个位置，并没有实际的意义。所以在解引用时，不能解它。

同样的，在使用迭代器操作 **string** 字符时，一定要检查字符是否为空。对于一个空字符，即容器为空，那么**begin** 和 **end** 指向的是同一个迭代器，示例程序如下：

```
int main() {
    string str = "asdfjkl";
    if (str.begin() != str.end())
    {
        auto it = str.begin();
        *it = toupper(*it);   //*it 表示解引用，和指针的接引用类似。
    }
    cout << str << endl;  // 输出结果为 Asdfjkl
}
```

如果把每个字符都变成大写，可以这样修改程序：**(了解)**

```
int main() {
    string str = "asdfjkl";
    for (auto it = str.begin(); it != str.end(); ++it)
    {
        *it = toupper(*it);
    }
    cout << str << endl;  // 输出ASDFJKL
}
```

进阶篇

1、取子字符串 substr()

s.substr(pos,n)返回一个 **string** ,包含 **s** 中从位置 **pos** 开始的 **n** 个字符。其中，**pos**的默认值是 **0, n**的默认值是 **size()-pos**。比如：

```
int main() {
    string str = "asdfjkl";
    string s1;
    string s2;
    string s3;
    s1 = str.substr(1, 4);   // 输出sdfj
    s2 = str.substr(1);     // 输出sdfjkl
    s3 = str.substr(10);     // 抛出异常
}
```

2、改变 string 的方法

(1) s.insert(pos, args) :在 s 的位置 pos 之前，插入args 指定的字符。其中pos可以为下标或者迭代器。insert()版本有很多，有迭代器版本，有下标版本，也有C风格字符串版本。而 args 具体指什么，详见下面程序：

注：argc 一般指(为列举包含字符数组的情况)：

str : 字符串str

str,pos,len : 字符串str 从位置pos开始长度为len

n,c : n个字符 c

```
int main() {
    string s = "abcde";
    string s1 = "xyz";
    string s2 = "xyz";
    string s3 = "xyz";
    string s4 = "xyz";
    //s.insert(0, s1);  // 在s 首元素之前，插入 s1 。输出结果为：xyzabcde
    //s.insert(2,s2,1,2);  // 输出 abyzcde，在s第二个位置之前,插入s2 下标从 1 到 2 的字符
    //s.insert(2, 5,'!');  // 输出 ab!!!!!cde。在 s 的第二个位置之前，插入5个单字符!
    //s.insert(2, 5, '!,!');  // 输出仍然是ab!!!!!cde，因为对于' '里面的只取第一个字符
    //s.insert(2, 5, "!,!");  报错，只能插入字符，不能插入字符串
    cout << s << endl;
}
```

(2) s.erase(pos,len):删除 s 中从 pos 开始的 len 个字符。如果省略 len ,则从pos开始删除到末尾。如果 pos 和 len 都省略，则将 s 都删除，返回空字符串，比如：

```
int main() {
    string s = "abcde";
    //s.erase();     //输出为空
    //s.erase(2);   // 输出ab
    s.erase(2,2);   // 输出 abe
    cout << s << endl;
}
```

(3) s.assign(args) : 将 s 中的字符替换为args指定的字符。(感觉没啥太大的用，类似于赋值吧)

```
int main() {
    string s = "abcde";
    string s1 = "xyz";
    s.assign(s1);
    cout << s << endl;  // 输出 xyc
}
```

(4) s.replace(range,args): 删除 range 范围内的字符，并把其替换为 argc 字符。其中，range 可以为一个下标或一个长度，或者一对指向 s 的迭代器

```
int main() {
    string s = "abcde";
    string s1 = "xyz";
    //s.replace(1,1,s1);  // 输出 abxyzde，也就是删除了 b，将 b 替换为xyz
    // s.replace(1, 1, "pp"); // 输出 appcde
    cout << s << endl;  //
}
```

(5)s.append(args) : 在 s 的末尾追加args，感觉和 s += args;

3、string 的搜索操作（返回下标，若未找到，就返回 npos）

先看几个函数，如果能灵活使用这几个函数，将大大的提高编程效率：

```
s.find(args) : 查找 args 第一次出现的位置
s.find_first_of(args) : 查找args任意一个字符第一次出现的位置
s.rfind(args) : 查找 args 最后一次出现的位置
s.find_last_of(args) : 查找args任意一个字符最后一次出现的位置
s.find_first_not_of(args) : 查找第一个不在args中的字符
s.find_last_not_of(args) : 查找最后一个不在args中的位置
```

其中，args 必须是下面四个之一：

```
c, pos    : 从 s 中位置 pos 开始查找字符 c
str, pos  :从 s 中位置 pos 开始查找字符串str
cp, pos   :从 s 中位置 pos 开始查找指针 cp 指向的以空字符串结尾的 C 风格字符串
cp,pos, n : 从 s 中位置 pos 开始查找指针 cp 指向的数组前 n 个字符
```

示例程序：

```
int main() {
    string s = "a,b,c,d,e";
    char *s1 = ",";
    char *s2 = "zx";
    int index = 0;
    //index = s.find(',');  // 下标数值为 1
    // index = s.find("b,");  // 下标为2
    //index = s.find(s1);  // 输出 为1
    index = s.find(s2);  // 输出为 -1，表示未找到
    cout << index << endl;
}
```

再有，

```
int main() {
    string s = "asdfjkl;qwertyuiop!@<,,";
    int index = s.find_first_of(',');  // 下标数值为 21。
    index = s.find_last_of(',');   // 下标数值为 22
    cout << index << endl;
}
```

注意：使用 **s.find_first_of(args)** 在 **s** 中寻找 **args** 任意一个字符第一次出现在 **s** 中的位置

```
int main() {
    string s = "pi=3.14159";
    int index = s.find_first_of("chghsf087632");  // 下标数值为 3 ，即 s[3] 是最早出现在"chghsf087632"里面的
    cout << index << endl;
}
```

4、数值转换

一般情况下，一个数的字符表示不同于其数值。数值 **15** 保存为 **16** 位的 **short** 类型，其二进制为 **0000000000001111**，而字符串 **"15"** 存为两个 **Latin-1** 编码的 **char** ,二进制为 **0011000100110101**，第一个字节表示字符**“1”**,其八进制值为 **061**,第二个字节表示**‘5’**,八进制值为 **65.**

注意：要转换为数值的 **string** 变量中的第一个非空白符必须是数值中可能出现的字符。比如：

```
int main() {
    int i = 15;
    string s = to_string(i);  // 将整数 i 转换为字符 表示形式，
    double b = stod(s);  // 将字符串转换成浮点数。
    cout << b << endl;   //  15
}
```

string 和字符之间的转换函数及其用法：

```
int main() {
    string s = "pi = 3.1415926";
    double pi = stod(s.substr(s.find_first_of("+-.0123456789")));
    cout << pi << endl;  // 3.14259
}
```