

C++ 字符串

C++ 提供了以下两种类型的字符串表示形式：

- C 风格字符串
- C++ 引入的 string 类类型

C 风格字符串

C 风格的字符串起源于 C 语言，并在 C++ 中继续得到支持。字符串实际上是使用 `null` 字符 `'\0'` 终止的一维字符数组。因此，一个以 `null` 结尾的字符串，包含了组成字符串的字符。

下面的声明和初始化创建了一个 "Hello" 字符串。由于在数组的末尾存储了空字符，所以字符数组的大小比单词 "Hello" 的字符数多一个。

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

依据数组初始化规则，您可以把上面的语句写成以下语句：

```
char greeting[] = "Hello";
```

以下是 C/C++ 中定义的字符串的内存表示：

Index	0	1	2	3	4	5
Variable	H	e	l	l	o	\0
Address	0x23451	0x23452	0x23453	0x23454	0x23455	0x23456

其实，您不需要把 `null` 字符放在字符串常量的末尾。C++ 编译器会在初始化数组时，自动把 `'\0'` 放在字符串的末尾。让我们尝试输出上面的字符串：

实例

```
#include <iostream>

using namespace std;

int main ()
{
    char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

    cout << "Greeting message: ";
    cout << greeting << endl;

    return 0;
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Greeting message: Hello
```

C++ 中有大量的函数用来操作以 `null` 结尾的字符串：supports a wide range of functions that manipulate null-terminated strings:

序号	函数 & 目的
1	strcpy(s1, s2); 复制字符串 s2 到字符串 s1。
2	strcat(s1, s2); 连接字符串 s2 到字符串 s1 的末尾。
3	strlen(s1); 返回字符串 s1 的长度。
4	strcmp(s1, s2); 如果 s1 和 s2 是相同的，则返回 0；如果 s1<s2 则返回值小于 0；如果 s1>s2 则返回值大于 0。
5	strchr(s1, ch); 返回一个指针，指向字符串 s1 中字符 ch 的第一次出现的位置。

6	<code>strstr(s1, s2);</code> 返回一个指针，指向字符串 s1 中字符串 s2 的第一次出现的位置。
---	--

下面的实例使用了上述的一些函数：

实例

```
#include <iostream>
#include <cstring>

using namespace std;

int main ()
{
    char str1[11] = "Hello";
    char str2[11] = "World";
    char str3[11];
    int len ;

    // 复制 str1 到 str3
    strcpy( str3, str1);
    cout << "strcpy( str3, str1) : " << str3 << endl;

    // 连接 str1 和 str2
    strcat( str1, str2);
    cout << "strcat( str1, str2): " << str1 << endl;

    // 连接后，str1 的总长度
    len = strlen(str1);
    cout << "strlen(str1) : " << len << endl;

    return 0;
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
strcpy( str3, str1) : Hello
strcat( str1, str2): HelloWorld
strlen(str1) : 10
```

C++ 中的 String 类

C++ 标准库提供了 **string** 类类型，支持上述所有的操作，另外还增加了其他更多的功能。我们将学习 C++ 标准库中的这个类，现在让我们先来看看下面这个实例：

现在您可能还无法透彻地理解这个实例，因为到目前为止我们还没有讨论类和对象。所以现在您可以只是粗略地看下这个实例，等理解了面向对象的概念之后再回头来理解这个实例。

实例

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

using namespace std;

int main ()
{
    string str1 = "Hello";
    string str2 = "World";
    string str3;
    int len ;

    // 复制 str1 到 str3
    str3 = str1;
    cout << "str3 : " << str3 << endl;

    // 连接 str1 和 str2
    str3 = str1 + str2;
    cout << "str1 + str2 : " << str3 << endl;

    // 连接后，str3 的总长度
    len = str3.size();
    cout << "str3.size() : " << len << endl;

    return 0;
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
str3 : Hello
str1 + str2 : HelloWorld
str3.size() : 10
```

string类提供了一系列针对字符串的操作，比如：

- 1. append() -- 在字符串的末尾添加字符
- 2. find() -- 在字符串中查找字符串
- 4. insert() -- 插入字符
- 5. length() -- 返回字符串的长度
- 6. replace() -- 替换字符串
- 7. substr() -- 返回某个子字符串
- 8. ...

下面是关于string类的实例：

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
    //定义一个string类对象
    string http = "www.runoob.com";

    //打印字符串长度
    cout<<http.length()<<endl;

    //拼接
    http.append("/C++");
    cout<<http<<endl; //打印结果为: www.runoob.com/C++

    //删除
    int pos = http.find("/C++"); //查找"C++"在字符串中的位置
    cout<<pos<<endl;
    http.replace(pos, 4, "");    //从位置pos开始，之后的4个字符替换为空，即删除
    cout<<http<<endl;

    //找子串runoob
    int first = http.find_first_of("."); //从头开始寻找字符'.'的位置
    int last = http.find_last_of(".");   //从尾开始寻找字符'.'的位置
    cout<<http.substr(first+1, last-first-1)<<endl; //提取"runoob"子串并打印

    return 0;
}
```

C++ 中输入的方式其实还有很多，下面来介绍一种与 C 语言中 getchar() 类似的。

```
cin.getline();
```

cin.getline() 是在输入一段字符完成后开始读取数据（注意，是输入完成后，以Enter为结束标志）

下面是一实例：输入一串字符，编程统计其中的数字个数和英文字母个数。输入的字符以 #为结束标志。

```
#include<iostream>
using namespace std;

#define N 100
int main()
{
    char X[N];
    cin.getline(X,N); //以cin.getLine形式输入
    int a=0,b=0;
    for(int i=0;i<N;i++)
    {
        if(X[i]=='#') //为#为结束标志
            break;
        if(X[i]>='0'&&X[i]<='9')
            a++; //统计数字个数
        if((X[i]>='a'&&X[i]<='z')||(X[i]>='A'&&X[i]<='Z'))
            b++; //统计英文字母个数
    }
    cout<<"数字个数: "<<a<<" 英文字母个数: "<<b<<endl;
    return 0;
}
```

```
        b++;                                //统计英文字母个数
    }
    cout<<a<<endl<<b<<endl;
    return 0;
}
```

字符串与vector

字符串字面值与标准库string不是同一种类型

```
string s("hello");
cout<<s.size()<<endl;           //OK
cout<<"hello".size()<<endl;     //ERROR
cout<<s+"world"<<endl;          //OK
cout<<"hello"+"world"<<endl;    //ERROR
```

strlen、sizeof与size()求字符串长度的区别

```
cout<<strlen("123")<<endl;      //返回 3
cout<<sizeof("123")<<endl;      //返回 4
string s = "123";
cout<<s.size()<<endl;           //返回 3
```

标准string库中的getline函数返回时会丢弃换行符const iterator与const_iterator的区别

```
vector<int>::const_iterator //不能改变指向的值，自身的值可以改变
const vector<int>::iterator //可以改变指向的值，自身的值不能改变
const vector<int>::const_iterator //自身的值和指向的值都是只读的
```

任何改变vector长度的操作都会使已存在的迭代器失效。如：在调用push_back之后，就不能再信赖指向vector的迭代器了

```
vector<int> ivec;
ivec.push_back(10);
vector<int>::iterator it = ivec.begin();
cout<<*it<<endl;
ivec.push_back(9);
cout<<*it<<endl;           //迭代器已经失效
```

之前一直搞不清 sizeof 和 strlen 到底该怎么区分，最近查了资料：

- 1、sizeof 操作符的结果类型是 size_t，它在头文件中 typedef 为 unsigned int 类型。该类型保证能容纳实现所建立的最大对象的字节大小。
- 2、sizeof 是运算符，strlen 是函数。
- 3、sizeof 可以用类型做参数，**strlen** 只能用 **char*** 做参数，且必须是以 **\0** 结尾的。

sizeof 还可以用函数做参数，比如：

```
short f();
printf("%d\n", sizeof(f()));
```

输出的结果是 sizeof(short)，即 2。

- 4、数组做 **sizeof** 的参数不退化，传递给 **strlen** 就退化为指针了。
- 5、大部分编译程序在编译的时候就把 **sizeof** 计算过了，是类型或是变量的长度，这就是 **sizeof(x)** 可以用来定义数组维数的原因。

```
char str[20]="0123456789";
int a=strlen(str); // a=10;
int b=sizeof(str); // 而 b=20;
```

- 6、strlen 的结果要在运行的时候才能计算出来，是用来计算字符串的长度，不是类型占内存的大小。
- 7、sizeof 后如果是类型必须加括弧，如果是变量名可以不加括弧。这是因为 **sizeof** 是个操作符不是个函数。
- 8、当适用一个结构类型或变量时，sizeof 返回实际的大小；当适用一静态地空间数组，sizeof 归还全部数组的尺寸；sizeof 操作符不能返回动态地被分派了的数组或外部的数组的尺寸。

数组作为参数传给函数时传的是指针而不是数组，传递的是数组的首地址，如：

```
fun(char [8])
fun(char [])
```

都等价于

```
fun(char *)
```

在 C++ 里参数传递数组永远都是传递指向数组首元素的指针，编译器不知道数组的大小。

如果想在函数内知道数组的大小，需要这样做：

进入函数后用memcpy拷贝出来，长度由另一个形参传进去

```
fun(unsigned char *p1, int len)
{
```

```
    unsigned char* buf = new unsigned char[len+1]
    memcpy(buf, p1, len);
}
```

看了上面的详细解释，发现两者的使用还是有区别的，从这个例子可以看得很清楚：

```
char str[20]="0123456789";
int a=strlen(str);          // a=10; >>>> strlen 计算字符串的长度，以结束符 0x00 为字符串结束。
int b=sizeof(str);         // 而 b=20; >>>> sizeof 计算的则是分配的数组 str[20] 所占的内存空间的大小，不受里面存储的内容改变。
```

上面是对静态数组处理的结果，如果是对指针，结果就不一样了。

```
char* ss = "0123456789";
sizeof(ss) 结果 4 ==》ss 是指向字符串常量的字符指针，sizeof 获得的是一个指针的之所占的空间,应该是长整型的，所以是 4。
sizeof(*ss) 结果 1 ==》*ss 是第一个字符 其实就是获得了字符串的第一位 '0' 所占的内存空间，是 char 类型的，占了 1 位
strlen(ss)= 10      ==》 如果要获得这个字符串的长度，则一定要使用 strlen。strlen 用来求字符串的长度；而 sizeof 是用来求指定变量或者变量类型等所占内存大小。
```

关于字符数组为什么可以以数组名来用cout输出数组内容，而普通数组不行。

先上范例：

```
int a[10] = {1,2,3,6,7};
char b[6] = {'h','a','p','p','y','\0'};
char c[] = "happy";
cout<<a<<endl;
cout<<b<<endl;
cout<<c<<endl;
```

输出结果为:

```
0x22fe6c
happy
happy
```

从以上范例可以看出，普通数组中以数组名用cout来输出，只会得到一串地址；用字符数组则会输出数组中的内容。

那为什么会这样呢？

答案：因为 char 型数组中的每一个元素都是一字节，所以每一个字符之间的地址都是 +1 的是连续的，所以当 cout 输出时读到字符数组中的 \0 便停止输出; 而 int 数组每个元素占 4 个字节所以数个数组中每个元素地址的间隔是 4，但其实它也是连续的，出现乱码是因没找到结束符。

Vs2017 使用 strcpy 的时候会报错，提示 strcpy 是不安全的，需要用 strcpy_s 代替。

```
#include "pch.h"
#include <iostream>
#include <cstring>

using namespace std;

int main()
{
    char str1[11] = "hello";
    char str2[12] = "world";
    char str3[11];
    int len;

    //复制str1到str2
    strcpy_s(str3, str1);
    cout << "strcpy(str3, str1):" << str3 << endl;

    return 0;
}
```

<cstring> 创建详解

```
#include<iostream>
#include<algorithm>
#include<cmath>
#include<cstring>
#include<cstdio>
using namespace std;
int main()
{
```

```
//1. 字符串的创建
cout<<"第一:字符串的创建!\n\n";
string a(4,'a');
cout<<"1.以 a 为原字符 4单位大小\n\n";
cout<<"string a(4,'a');\ncout<<a<<endl;\n";
cout<<a<<endl<<endl;

cout<<"2.任意大小的字符串\n\n";
cout<<"string b(\"bbbbbb\");\ncout<<b<<endl;\n";
string b("bbbbbb");
cout<<b<<endl<<endl;

cout<<"3.把某一字符串的某一部分\n(0位置开始4个长度)给c\n\n";
cout<<"string c(a,0,4) ;\ncout<<c<<endl;\n";
string c(a,0,4) ;
cout<<c<<endl<<endl;

cout<<"4. 10长度原长度不足补*;\n\n";
cout<<"c.resize(10,'*');\ncout<<c<<endl;\n";
c.resize(10,'*');
cout<<c<<endl<<endl;

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

<cstring> assign() 、 copy() 详解:

```
#include<iostream>
#include<algorithm>
#include<cmath>
#include<cstring>
#include<cstdio>
using namespace std;
int main()
{

cout<<"第二: 字符串的赋值 assign();"<<endl;
cout<<"1.感觉就像是append不过是抹除-覆盖\n";
cout<<"string e;\nchar f[10]=\"123456\"\ne.assign(f);\ne+=' ';\ncout<<e<<endl<<endl;\n";
string e;
char f[10]="123456";
e.assign(f);
e+=' ';
cout<<e<<endl<<endl;

cout<<"2.string区间 赋值都类似吧\n";
cout<<"e.assign(f,3,3);\ne+=' ';\ncout<<e<<endl<<endl;\ne.assign(f,3);\ncout<<e<<endl;\n";
e.assign(f,3,3);
e+=' ';
cout<<e<<endl;
e.assign(f,3);
cout<<e<<endl<<endl;

cout<<"3.某字符串char型 全部\n";
cout<<"char ssr[10]=\"asdqwezxc\";\ne.assign(ssr);\ncout<<ssr<<endl;\n";
char ssr[10]="asdqwezxc";
e.assign(ssr);
cout<<ssr<<endl<<endl;

cout<<"4.某字符串char型 前num个\n";
cout<<"e.assign(ssr,4);\ncout<<e<<endl;\n";
e.assign(ssr,4);
cout<<e<<endl<<endl;

cout<<"5.某字符赋值\n";
cout<<"赋值3个6\n";
e.assign(3,'6');
cout<<e<<endl<<endl;

cout<<"copy() 将d中的2位置开始的12个字符覆盖到char型数组ss上\n 必须为-> char型 <-否则报错";
```

```

cout<<" char ss[10]=\"123\";\n string dd;\nd.copy(ss,12,2);\ncout<<ss<<endl;\n";
char ss[15]="123";
string dd("abcdefghijklmn");
dd.copy(ss,12,2);
cout<<ss<<endl<<endl;

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

```

<cstring> append() 详解及其扩展(int, char):

```

#include<iostream>
#include<algorithm>
#include<cmath>
#include<cstring>
#include<cstdio>
using namespace std;
int main()
{
    cout<<"第三：字符串的添加与复制 append();\nstring d(a);\ncout<<d<<endl;\n//或者 d=d+a;/nd.append(b);\n";

    cout<<"1.在d的末尾添加字符串a\n\n";
    string d(a);
    d.append(b);
    cout<<d<<endl<<endl;

    cout<<"2.在d的末尾添加字符串/nb(0位置开始,2个长度)的数据\n\n";
    cout<<"d.append(b,0,2);\ncout<<d<<endl;\n";
    d.append(b,0,2);
    cout<<d<<endl<<endl;

    cout<<"3.添加4个 ~ 字符\n\n";
    cout<<"d.append(4, '~');\n";
    cout<<"cout<<d<<endl<<endl;\n";
    d.append(4, '~') ;
    cout<<d<<endl<<endl;

    system("pause");
    system("cls");

    cout<<"4. int 与 char 型添加 (好高兴自己想到的int ^_^ )\n";
    cout<<"char app[100]=\"aaabbb\";";
    cout<<"\nstring charr(\"-_-\");\n";
    cout<<"charr.append(app);\ncout<<charr<<endl\n";
    char app[100]="aaabbb";
    string charr("-_- ");
    charr.append(app);
    cout<<charr<<endl<<endl;

    cout<<"charr.append(app,4);\ncout<<charr<<endl<<endl;\n";
    charr.append(app,4);
    cout<<charr<<endl<<endl;

    cout<<"char型数组全部,char型数组的前4个\n\n****如果要添加中间***\n";
    cout<<"string tmp;\nstring tmp;tmp.assign(app);\ncharr.assign(\"\\\");\ncharr.append(tmp,1,4);\ncout<<charr<<endl<<endl;\n";
    string tmp;
    tmp.assign(app);
    charr.assign("");
    charr.append(tmp,1,4);
    cout<<charr<<endl<<endl;

    cout<<"5.int double 等等 通过 sprintf() <cstdio>作为转接\n";
    cout<<"int aaa=15314;\ndouble bbb=3.1415;\nchar aa[10];\nsprintf(aa, \"%d\", aaa);\ncharr.append(aa,0,4);\nsprintf(aa, \"%f\", bb
b);\ncharr.append(aa,0,4);\ncout<<charr<<endl;\n";
    int aaa=15314;
    double bbb=3.1415;
    char aa[10];
    sprintf(aa, "%d", aaa);
    charr.append(aa,0,4);
    sprintf(aa, "%f", bbb);
    charr.append(aa,0,4);
    cout<<charr<<endl<<endl;
}

```



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

C++ 中常见的几种输入字符串的方法如下：

cin、cin.get()、cin.getline()、getline()、gets()、getchar()

1. cin>>

用法一：最常用、最基本的用法，输入一个数字：

```
#include <iostream>
using namespace std;
int main ()
{
    int a,b;
    cin>>a>>b;
    cout<<a+b<<endl;
}

//输入：2[回车]3[回车]
//输出：5
```

用法二：接受一个字符串，遇“空格”、“Tab”、“回车”都结束

```
#include <iostream>
using namespace std;
int main ()
{
    char a[20];
    cin>>a;
    cout<<a<<endl;
}

//输入：jkljkljkl
//输出：jkljkljkl

//输入：jkljkl jkljkl //遇空格结束，所以不能输入多个单词
//输出：jkljkl
```

2. cin.get()

用法一：cin.get(字符变量名)可以用来接收字符

```
#include <iostream>
using namespace std;
int main ()
{
    char ch;
    ch=cin.get(); //或者cin.get(ch);只能获取一个字符
    cout<<ch<<endl;
}

//输入：jljkljkl
//输出：j
```

用法二：cin.get(字符数组名，接收字符数)用来接收一行字符串，可以接收空格

```
#include <iostream>
using namespace std;
int main ()
{
    char a[20];
    cin.get(a,20); //有些类似getline。可以输入多个单词，中间空格隔开。
    cout<<a<<endl;
}

//输入：jkl jkl jkl
//输出：jkl jkl jkl

//输入：abcdeabcdeabcdeabcdeabcde （输入25个字符）
//输出：abcdeabcdeabcdeabcd （接收19个字符+1个'\0'）
```


用法三：cin.get(无参数)没有参数主要是用于舍弃输入流中的不需要的字符, 或者舍弃回车, 弥补cin.get(字符数组名,接收字符数目)的不足.

```
#include <iostream>
using namespace std;

int main(void)
{

    char arr[10];
    cin.get(arr,10);
    cin.get();//用于吃掉回车，相当于getchar();
    cout<<arr<<endl;
    cin.get(arr,5);
    cout<<arr<<endl;
    system("pause");
    return 0;
}

//输入abcdefghi
//输出abcdefghi
//输入abcde
//输出abcd
//请按任意键继续
```

```
#include <iostream>
using namespace std;

int main(void)
{

    char arr[10];
    cin.get(arr,10);
    //cin.get();//用于吃掉回车，相当于getchar(); 现在把这行注释掉试试
    cout<<arr<<endl;
    cin.get(arr,5);
    cout<<arr<<endl;
    system("pause");
    return 0;
}

//输入abcdefghi
//输出abcdefghi
//请按任意键继续
```

3.cin.getline()

cin.getline(): 接受一个字符串，可以接收空格并输出

```
#include <iostream>
using namespace std;
int main ()
{
    char m[20];
    cin.getline(m,5); //与上面基本相同。
    cout<<m<<endl;
}

//输入: jkLjkLjkL
//输出: jkLj
```

接受5个字符到m中，其中最后一个为'\0'，所以只看到4个字符输出；

如果把5改成20：

```
输入: jkLjkLjkL
输出: jkLjkLjkL

输入: jkLf fjlsjf fjsdklf
输出: jkLf fjlsjf fjsdklf
```

延伸：

cin.getline()实际上有三个参数，cin.getline(接受字符串到m,接受个数5,结束字符)

当第三个参数省略时，系统默认为'\0' 是'/n'吧。

如果将例子中cin.getline()改为cin.getline(m,5,'a');当输入jlkjkljkl时输出jklj，输入jkaljkljkl时，输出jk

当用在多维数组中的时候，也可以用cin.getline(m[i],20)之类的用法：

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

```
using namespace std;

int main ()
{
    char m[3][20];
    for(int i=0;i<3;i++)
    {
        cout<<"\n请输入第"<<i+1<<"个字符串: "<<endl;
        cin.getline(m[i],20);
    }

    cout<<endl;
    for(int j=0;j<3;j++)
        cout<<"输出m["<<j<<"]的值:"<<m[j]<<endl;

}
```

测试：

```
请输入第1个字符串:
kskr1

请输入第2个字符串:
kskr2

请输入第3个字符串:
kskr3

输出m[0]的值:kskr1
输出m[1]的值:kskr2
输出m[2]的值:kskr3
```

4. getline()

getline()：接受一个字符串，可以接收空格并输出，需包含 #include<string>。

```
#include<iostream>
#include<string>
using namespace std;
int main ()
{
    string str;
    getline(cin,str);
    cout<<str<<endl;
}
```

测试：

```
输入: jk1jk1jk1 //VC6中有个bug,需要输入两次回车。
输出: jk1jk1jk1

输入: jk1 jfksldfj jklsjfl
输出: jk1 jfksldfj jklsjfl
```

和cin.getline()类似，但是cin.getline()属于istream流，而getline()属于string流，是不一样的两个函数

5. gets()

gets()：接受一个字符串，可以接收空格并输出，需包含 #include<string>。

```
#include<iostream>
#include<string>
using namespace std;
int main ()
{
    char m[20];
    gets(m); //不能写成m=gets();
    cout<<m<<endl;
}
```

测试：

```
输入: jk1jk1jk1
输出: jk1jk1jk1
```

```
输入: jkl jkl jkl
输出: jkl jkl jkl
```

类似cin.getline()里面的一个例子，gets()同样可以用在多维数组里面：

```
#include<iostream>
#include<string>
using namespace std;

int main ()
{
    char m[3][20];
    for(int i=0;i<3;i++)
    {
        cout<<"\n请输入第"<<i+1<<"个字符串: "<<endl;
        gets(m[i]);
    }

    cout<<endl;
    for(int j=0;j<3;j++)
        cout<<"输出m["<<j<<"]的值:"<<m[j]<<endl;
}
```

测试：

```
请输入第1个字符串:
kskr1

请输入第2个字符串:
kskr2

请输入第3个字符串:
kskr3

输出m[0]的值:kskr1
输出m[1]的值:kskr2
输出m[2]的值:kskr3
```

自我感觉gets()和cin.getline()的用法很类似，只不过cin.getline()多一个参数罢了；

这里顺带说明一下，对于本文中的这个kskr1,kskr2,kskr3的例子，对于cin>>也可以适用，原因是这里输入的没有空格，如果输入了空格，比如“ks kr jkl[回车]”那么cin就会已经接收到3个字符串，“ks,kr,jkl”；再如“kskr 1[回车]kskr 2[回车]”，那么则接收“kskr,1,kskr”；这不是我们所要的结果！而cin.getline()和gets()因为可以接收空格，所以不会产生这个错误；

6.getchar()

getchar()：接受一个字符，需包含 #include<string>。

```
#include<iostream>
using namespace std;
int main ()
{
    char ch;
    ch=getchar(); //不能写成getchar(ch);
    cout<<ch<<endl;
}
```

测试：

```
输入: jkljkljkl
输出: j
```

getchar()是C语言的函数，C++也可以兼容，但是尽量不用或少用；

12 关于整形数组的解释是错误的，即使整形数组加上结束符，cout输出的也是地址，这是c++的机制。只有数组是char类型时，cout数组名才会输出内容，其他类型的都会输出地址。

原始字符串(C++11)

R"(字符串)" 为默认格式的原始字符串，原始字符串内的字符不进行转义。

```
#include<iostream>
using namespace std;
int main ()
{
    cout << R"(原始\t字\n字符串)" << endl;
    return 0;
}
```

输出:

原始\t字\n字符串

自定义分隔符

分隔符可支持自定义，写在括号外部，左右需要一致，不是所有的字符都可以设置为分隔符：

```
#include<iostream>
using namespace std;
int main ()
{
    cout << R"##^*(1231)"123)##^&*" << endl;
    return 0;
}
```

输出:

1231)"123

字符串前缀：

L	：	类型	wchar_t	编码	：	Unicode
u	：	类型	char16_t	编码	：	UTF-16 (C++11)
U	：	类型	char32_t	编码	：	UTF-32 (C++11)
u8	：	类型	char	编码	：	UTF-8 (C++11)

实例：

```
#include<iostream>
using namespace std;
int main ()
{
    wcout << L"Unicode 字符串" << endl; //控制台字符类型为Unicode时，正常输出
    wcout << u"utf-16 字符串" << endl; //wcout 不支持的类型，输出地址
    wcout << U"utf-32 字符串" << endl; //wcout 不支持的类型，输出地址
    cout << u8"utf-8 字符串" << endl; //控制台字符类型为UTF-8时，正常输出
    return 0;
}
```

输出:

Unicode 字符串
00007FF7FB864480
00007FF7FB864498
utf-8 瀛樫涓?

字符串前缀与原始字符串组合：

```
#include<iostream>
using namespace std;
int main ()
{
    wcout << LR"(\t字\b符\r串\n)" << endl;
    return 0;
}
```

输出:

\t字\b符\r串\n