C++中的字符串流详解,_#include _一丁_的博客-CSDN博客

成就一亿技术人!

注：本文只详解C++中的#include <iostream> ,#include <sstream>,暂不详解#include <fstream>。等后期博主会补充。

与 C 类似，C++ 不具有内置输入/输出功能。 但是，所有 C++ 编译器都捆绑了一个系统的、面向对象的 I/O 包，称为 iostream 类。 该流是 iostream 类中的核心概念。 可将流对象视为一个智能文件，此文件充当字节的源和目标。 流的特征由其类和自定义的插入和提取运算符确定。

通过设备驱动程序，磁盘操作系统可将键盘、屏幕、打印机和通信端口作为扩展文件来处理。 iostream 类与这些扩展文件进行交互。 内置类支持使用与磁盘 I/O 相同的语法写入内存或从中读取，从而可以轻松派生流类。

目录

一、输出流：

输出流对象是字节的目标。 三个最重要的输出流类为 `ostream`、`ofstream` 和 `ostringstream`。

如果只使用预定义的 `cout`、`cerr` 或 `clog` 对象，则无需构造输出流对象，直接使用即可。 而对于输出文件流和输出字符串流对象需要使用构造函数构造出一个对象：

（1）构造输出流对象

输出文件流构造函数

可使用以下两种方式之一构造输出文件流：

①使用默认的构造函数，然后调用 open 成员函数。

```
ofstream myFile;
myFile.open("filename");


ofstream* pmyFile = new ofstream;
pmyFile->open("filename");
```

②在构造函数调用过程中指定文件名和模式标志。

```
ofstream myFile("filename", ios_base::out);
```

输出字符串流构造函数

若要构造一个输出字符串流，可以按照以下方法使用 ostringstream

default (1)	`explicit ostringstream (ios_base::openmode which = ios_base::out);`
initialization (2)	`explicit ostringstream (const string& str,` `ios_base::openmode which = ios_base::out);`

构造一个ostringstream对象:

①空构造函数(默认构造函数)

以空序列作为内容构造一个ostringstream对象。

在内部，它的ostream基构造函数被传递一个指向stringbuf对象的指针，该对象由基于自身的参数构造。

②初始化的构造函数

用str的副本作为内容构造一个ostringstream对象。

在内部，它的ostream基构造函数被传递一个指向stringbuf对象的指针，该对象由基于str的值构造，并将其作为参数。

```
// ostringstream constructor
#include <iostream>     // std::cout, std::ios
#include <sstream>      // std::ostringstream


in () {
::ostringstream foo;                            // out
::ostringstream bar (std::ostringstream::ate);  // out|ate,在末尾继续添加


.str("Test string");
.str("Test string");
```

```
<< 101;
<< 101;

::cout << foo.str() << '\n';
::cout << bar.str() << '\n';
rn 0;
}

//Output:
//101t string
//Test string101
```

注1：对输出流操作符' << '解释

在这里首先介绍我们常见的 ' cout<< '，因为 cout 为系统已经创建好的对象，所以我们可以直接使用，在使用的时候需要和输出流操作符捆绑在一起使用，而 cout 对象本身就有将内容打印的功能，即将输出流' << '后面的内容打印出来。例如 cout << "打印字符 A"; 在这其中流操作符' << '的作用就是就是将其后的内容拷贝到 cout 中，cout 实现打印功能。（可以在流操作符后面加一横线 ' <<- ' 表示将后面的内容赋予 cout，然后 cout 实现输出，当然这只是便于理解的一种说法）

同理对于我们创建的 ostringstream 对象 ' foo ' 输出流操作符' << '实现的功能同样是将后面的内容拷贝给' foo '（foo 为 ostringstream 创建的对象），可以理解为实现构造函数的功能。

注2：对std :: ostringstream :: str解释

```
 str（）const;
tr（const string＆s）;

/*==================
函数功能：获取/设置内容
第一种形式：返回一个带当前流内容副本的string对象。

第二种形式：将s设置为流的内容，丢弃任何以前的内容。对象保留其打开模式:如果其中包含ios_base::ate，则写入位置将移动到新序列的末尾。

在内部，函数调用其内部字符串缓冲区对象的str成员。

返回值：在流缓冲区中包含当前内容副本的字符串对象。
==================*/
```

```
//example
// ostringstream::rdbuf
#include <string>       // std::string
#include <iostream>     // std::cout
#include <sstream>      // std::ostringstream

in () {
::ostringstream oss;
<< "One hundred and one: " << 101;
::string s = oss.str();
::cout << s << '\n';
rn 0;
}

//output:
//One hundred and one: 101
```

（2）使用插入运算符并控制格式

输出宽度

若要对齐输出，您指定的输出宽度的每一项上来setw操控器在流中或通过调用width成员函数。 此示例右对齐至少为 10 个字符宽的列中的值：

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

```
int main( )
{
double values[] = { 1.23, 35.36, 653.7, 4358.24 };
for( int i = 0; i < 4; i++ )
   {
      cout.width(10);
      cout << values[i] << '\n';
   }
}
```


前导空白被添加到任何少于 10 个字符宽的值。

若要填充一个字段，请使用fill成员函数，设置具有指定的宽度的字段的填充字符的值。 默认值为空白。 若要填充带星号的数字的列，请修改以前的 for 循环，如下所示：

```
(int i = 0; i <4; i++)
{
ut.width(10);
ut.fill('*');
ut << values[i] << endl;
}
//output
//******1.23
//*****35.36
//*****653.7
//***4358.24
```

若要在同一行中指定数据元素的宽度，请使用 setw 操控器:

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

int main( )
{
double values[] = { 1.23, 35.36, 653.7, 4358.24 };
char *names[] = { "Zoot", "Jimmy", "Al", "Stan" };
for( int i = 0; i < 4; i++ )
      cout << setw( 7 )  << names[i]
           << setw( 10 ) << values[i] << endl;
}
```

width成员函数声明中<iostream >。 如果使用 setw 或任何其他带参数的操控器，则必须包含 <iomanip>。 在输出中，在宽度为 6 的字段中打印字符串，在宽度为 10 的字段中打印整数：

```
  Zoot      1.23
 Jimmy     35.36
    Al     653.7
  Stan   4358.24
```

setw和width都不截断值。如果格式化输出超过宽度，则打印整个值，这取决于流的精度设置。setw和width只影响当前字段。在打印一个字段之后，字段宽度恢复到其默认行为(必要的宽度)。但是，其他流格式选项在更改之前仍然有效。

对齐

输出流默认为右对齐文本。要左对齐上一个示例中的名称并右对齐数字，请按如下所示替换for循环：

```
for (int i = 0; i <4; i++)
    cout << setiosflags(ios::left)
```

```
        << setw(6) << names[i]
        << resetiosflags(ios::left)
        << setw(10) << values[i] << endl;
```

使用带有左枚举数的setiosflags操作器设置左对齐标志。这个枚举数是在ios类中定义的，所以它的引用必须包含ios::前缀。resetiosflags操作器关闭左对齐标志。与width和setw不同，setiosflags和resetiosflags的效果是永久性的。

精度

浮点精度的默认值为6。例如，数字3466.9768打印为3466.98。要更改此值的打印方式，请使用setprecision操纵器。操纵器有两个标志：fixed和 scientific。如果fixed设置，打印为3466.976800。如果scientific设置，则打印为3.4669773 + 003。

```
(int i = 0; i <4; i++)
ut << setiosflags(ios::left)
   << setw(6)
   << names[i]
   << resetiosflags(ios::left)
   << setw(10)
   << setprecision(1)
   << values[i]
   << endl;
//output
//Zoot          1
//Jimmy     4e+01
//Al        7e+02
//Stan      4e+03
```

要消除科学记数法，请在**for**循环之前插入此语句：

```
<< setiosflags(ios::fixed);


//output:
//Zoot         1.2
//Jimmy       35.4
//Al         653.7
//Stan      4358.2
```

将字符串插入到流中时，您可以通过调用stringstream::str()成员函数轻松地检索相同的字符串。但是，如果希望在稍后使用提取操作符将流插入到新字符串中，则可能会得到意想不到的结果，因为>>操作符在遇到第一个空白字符时将默认停止。

```
::stringstream ss;
::string inserted = "This is a sentence.";
::string extracted;


<< inserted;
>> extracted;


::cout << inserted;     //  This is a sentence.
::cout << extracted;    //  This
```

这种行为可以手动克服，但是为了使字符串往返更方便，c++ 14在<iomanip>中添加了std::quoted流操作器。插入时，quoted()用分隔符包围字符串(默认情况下是双引号'" '),提取时操作流提取所有字符，直到遇到最后一个分隔符。任何嵌入的引号都用转义字符转义(默认情况下为'\\')。

分隔符只出现在流对象中;它们不在提取的字符串中，而是在basic_stringstream::str返回的字符串中。

插入和提取操作的空格行为与字符串在代码中的表示方式无关，因此无论输入字符串是原始字符串文本还是常规字符串，引号操作符都是有用的。输入字符串，无论它的格式是什么，都可以嵌入引号、换行符、制表符等等，所有这些都将由quoted()操作器保存。

```
(std::string str) // or wstring
(const char* str) //or wchar_t*
(std::string str, char delimiter, char escape) // or wide versions
(const char* str, char delimiter, char escape) // or wide versions
```

二、输入流

输入流对象是字节的源。三个最重要的输入流类是istream、ifstream和istringstream。

istream类最适合用于顺序的文本模式输入。您可以为已缓存或未缓存操作配置istream类的对象。基类ios的所有功能都包含在istream中。您很少从istream类构造对象。相反，您通常将使用预定义的cin对象，它实际上是ostream类的一个对象。在某些情况下，可以在程序启动后将cin分配给其他流对象。

ifstream类支持磁盘文件输入。如果需要一个只输入的磁盘文件，构造一个类ifstream的对象。您可以指定二进制或文本模式数据。如果在构造函数中指定文件名，则在构造对象时将自动打开文件。否则，您可以在调用默认构造函数后使用open函数。许多格式化选项和成员函数都适用于ifstream对象。基本类ios和istream的所有功能都包含在ifstream中。

与库函数sscanf_s类似，istringstream类支持来自内存字符串的输入。要从具有空终止符的字符数组中提取数据，分配并初始化字符串，然后构造istringstream类的对象。

（1）构造输入流对象

如果仅使用cin对象，则无需构造输入流。如果使用ifstream，istringstream则必须构造输入流：

（1）输入文件流构造函数

有两种方法可以创建输入文件流：

①使用void参数构造函数，然后调用open成员函数：

```
am myFile; // On the stack

.open("filename");


am* pmyFile = new ifstream; // On the heap

e->open("filename");
```

②在构造函数调用中指定文件名和模式标志，从而在构造过程中打开文件：

```
ifstream myFile("filename");
```

（2）输入字符串流构造函数

输入字符串流构造函数需要预先分配的预初始化存储的地址：

```
default (1)

explicit istringstream (ios_base::openmode which = ios_base::in);

initialization (2)

explicit istringstream (const string& str,

                        ios_base::openmode which = ios_base::in);
```

①空构造函数（默认构造函数）

构造一个 istringstream具有空序列作为内容的对象。

在内部，它的istream基构造函数被传递一个指向stringbuf对象的指针，该对象由一个基于该对象的参数构造。

②初始化的构造函数

用str的副本作为内容构造一个istringstream对象。

在内部，它的istream基构造函数被传递一个指向stringbuf对象的指针，该对象由基于str和which的参数构造。

```
// istringstream constructors.

#include <iostream>     // std::cout

#include <sstream>      // std::istringstream

#include <string>       // std::string


in () {


::string stringvalues = "125 320 512 750 333";

::istringstream iss (stringvalues);


(int n=0; n<5; n++)

{

t val;

s >> val;

d::cout << val*2 << '\n';

}


rn 0;

}


//output:

//250

//640

//1024

//1500

//666
```

③std::istringstream::str()解释

```
string str（）const;
void str（const string&s）;
```

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

int main () {
  std::istringstream iss;
  std::string strvalues = "32 240 2 1450";

  iss.str (strvalues);

for (int n=0; n<4; n++)
  {
int val;
    iss >> val;
    std::cout << val << '\n';
  }
  std::cout << "Finished writing the numbers in: ";
  std::cout << iss.str() << '\n';
return 0;
}
```

三、格式转换

string到int的转换

```
 ss = "10000";
= 0;
 << ss;
 >> n;              //n等于10000
```

重复利用stringstream对象

如果你打算在多次转换中使用同一个stringstream对象，则每次转换前要使用clear()；这样能提高程序运行效率。

在类型转换中使用模板

定义函数模板来将一个任意的类型转换到特定的目标类型。例如，需要将各种数字值，如int、long、double转换成字符串，要使用以一个string类型和一个任意值t为参数的to_string()函数。to_string()函数将t转换为字符串并写入result中。使用str()成员函数来获取流内部缓冲的一份拷贝：

```
o_string(string & result,const T& t)
{
tringstream oss;       //创建一个输出流
```

```
s << t;                //把值传递如流中
sult = oss.str();      //获取转换后的字符转并将其写入result
}


//使用
_string(s1,10.5);  //double到string
_string(s2,123);   //int到string
_string(s3,true);  //bool到string
```

格式转换内容可参考：

https://www.cnblogs.com/wyuzl/p/6135537.html

sstring 可参考：https://blog.csdn.net/xw20084898/article/details/21939811