第四部分

标准 C++ 类库

标准C++定义了广泛的类集，提供对一些常见的活动、包括I/O字符串和数字处理的支持。类库不同于第三部分中描述的函数库。类库形成了C++语言的主要部分，同时定义了它的许多特征。虽然类库很大，但是它很容易掌握，因为它是按照面向对象的原则进行组织的。

标准C++库相当大，因此深层次地描述它的所有类、特征、属性和实现细节超出了本书的范围（对类库的全面描述需要整整一厚本书）。然而，虽然类库中有很大一部分是通用的，但是其中的某些则主要是让编译器开发人员或实现扩展或增强功能的程序员使用的。因此，本节只描述类库中用于一般的应用程序的部分，如果读者为特殊目的而使用类库，需要阅读关于C++标准的书，那里会包含类库的技术描述。

第32章　标准C++ I/O类

本章描述标准C++的I/O类库。正如在第二部分中解释的，现在常用的有两种版本的C++ I/O库。第一种是老式的库，不被标准C++所定义。第二种是现代的、模板化的标准C++ I/O系统。因为现代I/O库从本质上讲是老式库的超集，所以本章仅描述现代库。然而，绝大多数信息仍然适用于老的版本。

注意： 关于C++ I/O的概述，请参见第20章和第21章。

32.1　I/O类

标准C++ I/O系统是从一个较复杂的模板类系统构造成的，这些类如下表所示。

类	用途
basic_ios	提供一般用途的I/O操作
basic_streambuf	对I/O的低级支持
basic_istream	对输入操作的支持
basic_ostream	对输出操作的支持
basic_iostream	对输入/输出操作的支持
basic_filebuf	对文件I/O的低级支持
basic_ifstream	对文件输入的支持
basic_ofstream	对文件输出的支持
basic_fstream	对文件输入/输出的支持
basic_stringbuf	对基于字符串的I/O的低级支持
basic_istringstream	对基于字符串的输入的支持
basic_ostringstream	对基于字符串的输出的支持
basic_stringstream	对基于字符串的输入/输出的支持

I/O类层次的另一个部分是非模板类ios_base，它提供了对I/O系统各种元素的定义。

C++I/O系统是建立在两个相互关联、但不同的模板类层次上的。第一个是从称为basic_streambuf的低级I/O类中派生的。这个类提供基本的低级输入和输出操作，并且提供了对整个C++ I/O系统的基础支持。类basic_filebuf和basic_stringbuf是从basic_streambuf中派生的。除非正在从事高级I/O编程工作，否则不需要直接使用basic_streambuf或它的子类。

最常使用的类层次是从basic_ios中派生的。这是一个高级I/O类，提供了格式化、错误检查和与流I/O有关的状态信息。basic_ios用做几个派生类，包括basic_istream, basic_ostream和basic_iostream的基类。这些类被用来创建能够分别进行输入、输出、输入/输出操作的流。特别要指出的是，从basic_istream可派生出类basic_ifstream和basic_istringstream，从basic_ostream中可派生出类basic_ofstream和basic_ostringstream，从basic_iostream中可派生出basic_fstream和basic_stringstream。basic_ios的基类是ios_base。因此，从basic_ios派生出的任何类都可以访问ios_base的成员。

对于I/O类所操作的字符类型和与那些字符相关联的性质，I/O类都被参数化了。例如，下面是basic_ios的模板规范：

```
template <class CharType, class Attr = char_traits<CharType> >
  class basic_ios: public ios_base
```

其中，CharType指定了字符类型（例如char或wchar_t），Attr指定了描述它的属性的类型。通用类型的char_traits是一个定义与字符相关联的属性的实用类。

正如在第20章所解释的，I/O库创建了刚刚描述的模板类的两个规范：一个用于8位字符，另一个用于宽字符。下面是一个模板类名和它们的字符及宽字符之间映射的完整列表。

模板类	基于字符的类	基于宽字符的类
basic_ios	ios	wios
basic_istream	istream	wistream
basic_ostream	ostream	wostream
basic_iostream	iostream	wiostream
basic_ifstream	ifstream	wifstream
basic_ofstream	ofstream	wofstream
basic_fstream	fstream	wfstream
basic_istringstream	istringstream	wistringstream
basic_ostringstream	ostringstream	wostringstream
basic_stringstream	stringstream	wstringstream
basic_streambuf	streambuf	wstreambuf
basic_filebuf	filebuf	wfilebuf
basic_stringbuf	stringbuf	wstringbuf

因为绝大多数程序员使用基于字符的I/O，所以那些是本章使用的名称。因此，当谈到I/O类时，我们简单地使用它们的基于字符的名称，而不是它们的内部的模板名称。例如，本章将使用名称ios而不是basic_ios，使用istream而不是basic_istream，使用fstream而不是basic_fstream。记住，宽字符流也存在类似的类，它们的工作方式与这里所述的那些相同。

32.2 I/O头文件

标准C++ I/O系统依赖几个头文件，如下表所示：

头文件	用于
<fstream>	文件I/O
<iomanip>	带参数的I/O操作算子
<ios>	基本I/O支持
<iosfwd>	转发（forward）由I/O系统所用的声明
<iostream>	通用的I/O
<istream>	基本输入支持
<ostream>	基本输出支持
<sstream>	基于字符串的流
<streambuf>	低级I/O支持

这些头文件中有几个是I/O系统内部使用的。一般来讲，程序仅包括<iostream>, <fstream>, <sstream> 或<iomanip>。

32.3 格式化标记和 I/O 操作算子

每个流都与一组格式标记相关联，这组格式标记控制着信息被格式化的方式。ios_base 类声明了一个位掩码枚举 fmtflags，其中定义了下列值。

adjustfield	basefield	boolalpha	dec
fixed	floatfield	hex	internal
left	oct	right	scientific
showbase	showpoint	showpos	skipws
unitbuf	uppercase		

使用函数，如 setf( )和 unsetf( )，并使用这些值来设置或清除格式标记。在第 20 章中可以找到关于这些标记的详细描述。

除了直接设置或清除格式标记外，可以通过使用称为操作算子的特殊函数，来改变一个流的格式参数，其中的操作算子包含于一个 I/O 表达式中。标准操作算子显示于下表中：

操作算子	用途	输入 / 输出
boolalpha	打开 boolapha 标记	Input/Output
dec	打开 dec 标记	Input/Output
endl	输出一个换行符并清空流	Output
ends	输出一个空值	Output
fixed	打开 fixed 标记	Output
flush	清空一个流	Output
hex	打开 hex 标记	Input/Output
internal	打开 internal 标记	Output
left	打开 left 标记	Output
noboolalpha	关闭 boolalpha 标记	Input/Output
noshowbase	关闭 showbase 标记	Output
noshowpoint	关闭 showpoint 标记	Output
noshowpos	关闭 showpos 标记	Output
noskipws	关闭 skipws 标记	Input
nounitbuf	关闭 unitbuf 标记	Output
nouppercase	关闭 uppercase 标记	Output
oct	打开 oct 标记	Input/Output
resetiosflags (fmtflags *f*)	关闭在 f 中指定的标记	Input/Output
right	打开 right 标记	Output
scientific	打开 scientific 标记	Output
setbase(int *base*)	设置数的底为 base	Input/Output
setfill(int *ch*)	设置填充字符到 ch 中	Output
setiosflags(fmtflags *f*)	打开在 f 中指定的标记	Input/output
setprecision (int *p*)	设置精度的位数	Output
setw(int *w*)	设置域宽为 w	Output
showbase	打开 showbase 标记	Output

（续表）

操作算子	用途	输入/输出
showpoint	打开 showpoint 标记	Output
showpos	打开 showpos 标记	Output
skipws	打开 skipws 标记	Input
unitbuf	打开 unitbuf 标记	Output
uppercase	打开 uppercase 标记	Output
ws	跳过前面的空格	Input

要使用带一个参数的操作算子，必须包含<iomanip>。

32.4　几个数据类型

除了刚刚描述的 fmtflags 类型外，标准 C++ I/O 系统定义了其他几种类型。

32.4.1　streamsize 和 streamoff 类型

streamsize类型的对象能够容纳在任何一个I/O操作中传输的最大字节数，它通常是某种类型的整数。streamoff类型的对象能够容纳一个值，该值指示在流中的偏移位置，它通常是某种类型的整数。这些类型在头文件<ios>中定义，此头文件是 I/O 系统自动包含的。

32.4.2　streampos 和 wstreampos 类型

streampos类型的对象能够容纳一个值，该值表示在一个char流内的位置。wstreampos类型能够容纳一个值，该值表示在wchar_t流中的一个位置。这些是在<iosfwd>定义的，<iosfwd>是I/O系统自动包含的。

32.4.3　pos_type 和 off_type 类型

pos_type和off_type类型创建能够容纳一个值的对象（通常是整数），其中的值分别表示一个流中的位置和偏移量。这些类型是ios（和其他类）定义的，本质上与streamoff和streampos（或它们的宽字符对应物）一样。

32.4.4　openmode 类型

openmode类型是由 ios_base 定义的，描述了文件是怎样打开的，它将是下列值之一。

app	加到文件的末尾
ate	在创建时查找文件的末尾
binary	打开文件，进行二进制操作
in	打开文件，用于输入
out	打开文件，用于输出
trunc	清除以前存在的文件

通过把这些值“或”在一起，可以把它们组合起来。

32.4.5　iostate 类型

I/O流的当前状态是由一个iostate类型的对象描述的，它是由ios_base定义的一个枚举值，

ios_base包括下列成员。

名称	意义
goodbit	没有出现错误
eofbit	遇到文件末尾
failbit	出现一个非致命I/O错误
badbit	出现一个致命I/O错误

32.4.6 seekdir类型

seekdir类型描述了一个随机访问文件操作是怎样发生的。它是在ios_base内定义的，它的有效值如下所示。

beg	文件开始处
cur	当前位置
end	文件末尾

32.4.7 failure类

在ios_base中，定义了异常类型failure。它可以用做被I/O系统抛出的异常类型的基类。它继承了exception（标准异常类）。failure类有下面的构造函数。

```
explicit failure(const string &str);
```

其中，str是描述错误的消息。这个消息可从一个failure对象中获得，方法是调用它的what()函数，如下所示：

```
virtual const char *what( ) const throw( );
```

32.5 重载<<和>>运算符

下面的类重载与所有的内嵌数据类型有关的<<和/或>>运算符。

```
basic_istream
basic_ostream
basic_iostream
```

从这些类中派生的任何类都继承了这些运算符。

32.6 通用的I/O函数

本章其余部分描述了标准C++提供的通用的I/O函数。正如所解释的，标准C++ I/O系统是建立在一个复杂的模板类层次上的。应用程序编程时并不使用其中的许多低级类成员。因此，这里不讲述它们。

32.6.1 bad

```
#include <iostream>
bool bad() const;
```

bad( )函数是ios的一个成员。

如果在一个相关联的流中出现了一个致命的I/O错误，bad( )函数返回true。否则，返回false。

相关函数是 good( )。

32.6.2 clear

```
#include <iostream>
void clear(iostate flags = goodbit);
```

clear( )函数是 ios 的一个成员。

clear( )函数清除与一个流相关联的状态标记。如果 flags 是 goodbit（就像在默认时那样），那么清除所有的错误标记（重置为 0）。否则，状态标记将被重置为 flags 中指定的值。

相关函数是 rdstate( )。

32.6.3 eof

```
#include <iostream>
bool eof() const;
```

eof( )函数是 ios 的一个成员。

当遇到相关输入文件的末尾时，eof( )函数返回 true。否则，它返回 false。

相关函数是 bad( ), fail( ), good( ), rdstate( )和 clear( )。

32.6.4 exceptions

```
#include <iostream>
iostate exceptions() const;
void exceptions(iostate flags);
```

exceptions( )函数是 ios 的一个成员。

第一种形式返回一个 iostate 对象，指示哪个标记引起了异常。第二种形式设置这些值。

相关函数是 rdstate( )。

32.6.5 fail

```
#include <iostream>
bool fail() const;
```

fail( )函数是 ios 的一个成员。

如果在相关流中出现一个 I/O 错误，fail( )函数返回 true。否则，它返回 false。

相关函数是 good( ), eof( ), bad( ), clear( )和 rdstate( )。

32.6.6 fill

```
#include <iostream>
char fill() const;
char fill(char ch);
```

fill( )函数是 ios 的一个成员。

默认时，当需要填充一个域时，使用空格填充。然而，可以使用 fill( )函数指定填充字符并在 ch 中指定新的填充字符。返回老的填充字符。

要获得当前的填充字符，使用第一种形式的 fill( )，它返回当前的填充字符。

相关函数是 precision( )和 width( )。

32.6.7 flags

```
#include <iostream>
fmtflags flags() const;
fmtflags flags(fmtflags f);
```

flags( )函数是 ios (从 ios_base 继承而来)的一个成员。

第一种形式的 flags( )简单地返回相关流的当前格式标记设置。

第二种形式的 flags( ) 把与一个流相关联的所有格式标记设置为 f 指定的那样。当使用这种形式时，在 f 中发现的位模式被复制到与该流相关的格式标记中。这种形式也返回以前的设置。

相关函数是 unsetf( )和 setf( )。

32.6.8 flush

```
#include <iostream>
ostream &flush();
```

flush( )函数是 ostream 的一个成员。

flush( )函数使得与相关输出流相连的缓存被物理地写到这个设备上。函数返回到它的相关流的一个引用。

相关函数是 put( )和 write( )。

32.6.9 fstream, ifstream 和 ofstream

```
#include <fstream>
fstream();
explicit fstream(const char *filename,
                 ios::openmode mode = ios::in | ios::out);
ifstream();
explicit ifstream(const char *filename, ios::openmode mode=ios::in);

ofstream();
explicit ofstream(const char *filename,
                  ios::openmode mode=ios::out);
```

fstream( ), ifstream( )和 ofstream( )函数分别是 fstream, ifstream 和 ofstream 类的构造函数。

不带参数的 fstream( ), ifstream( )和 ofstream( )创建一个不与任何文件相关联的流。使用 open( )，可以把这个流链接到一个文件上。

带一个文件名作为第一个参数的fstream( ), ifstream( )和ofstream( )最常用于应用程序中。尽管使用 open( )函数打开一个文件很合适，大多数情况下，我们不这样做，因为当创建流时，这些ifstream, ofstream,和fstream构造函数自动打开这个文件。这些构造函数与 open( )函数有同样的参数和默认值（详细内容请参见 open)。例如，下面是最常见的打开文件的方式：

```
ifstream mystream("myfile");
```

如果由于某种原因不能打开文件，相关流变量的值将为 false。因此，不管是用一个构造函数来打开文件，还是通过显式调用 open( )来打开文件，你都会想要证实文件已经打开，可以通

过测试流的值来证实这一点。

相关函数是 close( )和 open( )。

32.6.10　gcount

```
#include <iostream>
streamsize gcount() const;
```

gcount( )函数是 istream 的一个成员。

gcount( )函数返回上一次输入操作所读取的字符数。

相关函数是 get( ), getline( )和 read( )。

32.6.11　get

```
#include <iostream>
int get();
istream &get(char &ch);
istream &get(char *buf, streamsize num);
istream &get(char *buf, streamsize num, char delim);
istream &get(streambuf &buf);
istream &get(streambuf &buf, char delim);
```

get( )函数是 istream 的一个成员。

通常，get( )读取输入流中的字符。无参数的 get( )读取相关流中的一个字符并返回那个值。

get(char &ch) 读取相关流中的字符并把那个值放到 ch 中。它返回那个流的一个引用。

get(char *buf, streamsize num)把字符读到由 buf 所指的数组中，直到读取 num1 个字符、或者发现一个新行、或者遇到文件末尾为止。由 buf 所指向的数组将被 get( )以 null 字符终止。如果在输入流中遇到换行符，它不被提取。相反，它保留在流中，直到下一个输入操作。这个函数返回该流的一个引用。

get(char *buf, streamsize num, char delim)把字符读到由 buf 所指的数组中，直到读取了 num-1 个字符、找到由 delim 所指定的字符、或者遇到文件末尾。由 buf 所指向的数组将被 get( )以 null 字符终止。如果在输入流中遇到分界符（delimiter character），它不被提取，相反，它保留在流中，直到下一个输入操作。这个函数返回一个该流的引用。

get(streambuf &buf)把输入流中的字符读到 streambuf 对象中。字符被读取，直到发现了一个换行符、或者遇到了文件末尾。它返回一个该流的引用。如果在输入流中遇到换行符，它不被提取。

get(streambuf &buf, char delim)把输入流中的字符读到 streambuf 对象中。字符被读取，直到发现由 delim 所指定的字符、或者遇到了文件末尾。它返回一个到该流的引用。如果在输入流中遇到了分界符，它不被提取。

相关函数是 put( ), read( )和 getline( )。

32.6.12　getline

```
#include <iostream>
istream &getline(char *buf, streamsize num);
istream &getline(char *buf, streamsize num, char delim);
```

getline( )函数是 istream 的一个成员。

getline(char *buf, streamsize num)把字符读到由 buf 指向的数组中，直到读取了 num-1 个字符、发现一个换行符、或者遇到文件末尾。由 buf 所指向的数组将被 getline( )以 null 字符终止。如果在输入流中遇到换行符，它被提取，但是不放到 buf 中。这个函数返回到该流的一个引用。

getline(char *buf, streamsize num, char delim)把字符读到由 buf 所指向的数组中，直到读取了 num-1 个字符、发现由 delim 所指的字符、或者遇到了文件末尾。由 buf 所指向的数组将被 getline( )以 null 字符终止。如果在输入流中遇到分界符，它被提取，但不放到 buf 中。这个函数返回到该流的一个引用。

相关函数是 get( )和 read( )。

32.6.13 good

```
#include <iostream>
bool good() const;
```

good( )函数是 ios 的一个成员。

如果在相关流的流中没有出现 I/O 错误，good( )函数返回 true。否则，返回 false。

相关函数是 bad( ), fail( ), eof( ), clear( )和 rdstate( )。

32.6.14 ignore

```
#include <iostream>
istream &ignore(streamsize num = 1, int delim = EOF);
```

ignore( )函数是 istream 的一个成员。

可以使用 ignore( )成员函数来读取和丢弃输入流中的字符。它读取并丢弃字符，直到忽略了 num 个字符（默认时为 1）、或者遇到由 delim 指定的字符（默认时为 EOF）。如果遇到了分界符，把它从输入流中删除。这个函数返回到该流的一个引用。

相关函数是 get( )和 getline( )。

32.6.15 open

```
#include <fstream>
void fstream::open(const char *filename,
                   ios::openmode mode = ios::in | ios:: out);
void ifstream::open(const char *filename,
                   ios::openmode mode = ios::in);
void ofstream::open(const char *filename,
                   ios::openmode mode = ios:: out | ios::trunc);
```

open( )函数是 fstream, ifstream 和 ofstream 的一个成员。

通过使用 open( )函数，可以把一个文件和一个流关联起来。这里，filename 是文件的名称，它可以包含一个路径标识符。mode 的值决定了如何打开这个文件。它必须是下面这些值之一（也可以是多个）：

ios::app
ios::ate
ios::binary

ios::in
ios::out
ios::trunc

可以把这些值中的两个或更多个组合起来，方法是把它们“或”在一起。

包含 ios::app 会引起所有到那个文件的输出被追加到末尾。这个值仅能和具有输出能力的文件一起使用。包含 ios::ate 会在文件打开时引起出现一个到文件末尾的查找。尽管 ios::ate 引起了一个到文件末尾的查找，I/O 操作仍然可以出现在文件中的任何地方。

ios::binary 值引起文件为进行二进制I/O操作而打开。默认时，文件是在文本模式中打开的。

ios::in 值指定文件能够输入。ios::out 值指定文件能够输出。然而，创建一个 ifstream 流就意味着输入，创建一个 ofstream 流意味着输出，使用 fstream 打开一个文件意味着输入和输出。

ios::trunc 值引起具有同一名称的以前存在的文件的内容被销毁，文件被截短到 0 长度。

总之，如果 open( )失败，该流为 false。因此，在使用一个文件前，应该试验以确保打开操作成功进行。

相关函数是 close( ), fstream( ), ifstream( )和 ofstream( )。

32.6.16　peek

```
#include <iostream>
int peek();
```

peek( )函数是 istream 的一个成员。

如果遇到文件末尾，peek( )函数返回流中的下一个字符或者 EOF。无论如何，它都不会把该字符从流中删除。

相关函数是 get( )。

32.6.17　precision

```
#include <iostream>
streamsize precision() const;
streamsize precision(streamsize p);
```

precision( )函数是 ios（从 ios_base 中继承而来）的一个成员。

默认时，当输出的是浮点值时，显示 6 位精度。然而，使用第二种形式的 precision( )，可以把这个数设置为 p 中规定的值。返回初始值。

第一种形式的 precision( )返回当前值。

相关函数是 width( )和 fill( )。

32.6.18　put

```
#include <iostream>
ostream &put(char ch);
```

put( )函数是 ostream 的一个成员。

put( )函数把 ch 写到相关联的输出流中。它返回到该流的一个引用。

相关函数是 write( )和 get( )。

32.6.19 putback

```
#include <iostream>
istream &putback(char ch);
```

putback( )函数是istream的一个成员。

putback( )函数把ch返回到相关联的输入流中。

相关函数是peek( )。

32.6.20 rdstate

```
#include <iostream>
iostate rdstate() const;
```

rdstate( )函数是ios的一个成员。

rdstate( )函数返回关联流的状态。C++ I/O系统维护关于与每个活动流（active stream）有关的每个I/O操作的结果的状态信息。流的当前状态保存于iostate类型的对象中，其中定义了下列标记：

名称	意义
goodbit	没有出现错误
eofbit	遇到文件末尾
failbit	出现一个非致命的I/O错误
badbit	出现一个致命的I/O错误

这些标记在ios (通过ios_base)内列出。

当没有出现错误时，rdstate( )返回goodbit。否则，设置一个错误位。

相关函数是eof( ), good( ), bad( ), clear( ), setstate( )和fail( )。

32.6.21 read

```
#include <iostream>
istream &read(char *buf, streamsize num);
```

read( )函数是istream的一个成员。

read( )函数从相关流中读入num个字节并把它们放到buf所指的缓存中。如果在读完num个字符之前到达了文件末尾，则read( )停止，设置failbit，并且缓存中包含能得到的所有字符（参见gcount( )）。read( )返回到该流的引用。

相关函数是gcount( ), readsome( ), get( ), getline( )和write( )。

32.6.22 readsome

```
#include <iostream>
streamsize readsome(char *buf, streamsize num);
```

readsome( )函数是istream的一个成员。

readsome( )函数从相关联的输入流中读入num个字节并把它们放到buf所指的缓存中。如果该流包含的字符少于num个，就读取那个数目的字符。readsome( )返回所读的字符数。read( )和readsome( )之间的区别是：如果有少于num个字符可得，readsome( )不设置failbit。

相关函数是 gcount( ), read( )和 write( )。

32.6.23　seekg 和 seekp

```
#include <iostream>
istream &seekg(off_type offset, ios::seekdir origin)
istream &seekg(pos_type position);

ostream &seekp(off_type offset, ios::seekdir origin);
ostream &seekp(pos_type position);
```

seekg( )函数是 istream 的一个成员，seekp( )函数是 ostream 的一个成员。

在 C++ 的 I/O 系统中，使用 seekg( )和 seekp( )函数来执行随机访问。要做到这一点，C++ I/O 系统管理两个与一个文件相关联的指针。一个是获取指针（get pointer），它指定下一个输入操作将出现在文件中的什么地方。另一个是放置指针（put pointer），它指定下一个输出操作将出现在文件中的什么地方。每次发生输入或输出操作时，自动按序给相应的指针加1。然而，使用 seekg( )和 seekp( )函数，可以以非顺序的方式访问文件。

带两个参数的 seekg( )形式把获取指针从 origin 指定的位置移动 offse 字节数。带两个参数的 seekp( ) 形式把放置指针从 origin 指定的位置移动 offset 字节数。offset 参数是 off_type 类型的，它能够包含 offset 具有的最大有效值。

origin 参数是 seekdir 类型的，且是一个具有这些值的枚举：

ios::beg	从头开始查找
ios::cur	从当前位置开始查找
ios::end	从末尾处开始查找

带一个参数的 seekg( )和 seekp( )把文件指针移动到由 position 指定的位置。分别使用对 tellg( ) 或 tellp( )的一个调用，一定可以获得这个值。pos_type 是一种能够包含 position 具有的最大有效值的类型。这些函数返回到相关联的流的引用。

相关函数是 tellg( )和 tellp( )。

32.6.24　setf

```
#include <iostream>
fmtflags setf(fmtflags flags);
fmtflags setf(fmtflags flags1, fmtflags flags2);
```

setf( )函数是 ios (从 ios_base 继承而来)的一个成员。

setf( )函数设置与一个流相关联的格式标记。参见本节前面关于格式标记的讨论。

第一种形式的 setf( )打开由 flags 指定的格式标记（所有其他标记不受影响）。例如，要为 cout 打开 showpos 标记，可以使用下面这条语句：

```
cout.setf(ios::showpos);
```

当想设置一个以上的标记时，可以把想设置的标记值或（OR）在一起。

重要的是要理解对 setf( )的调用是与一个特定的流相关的。不存在调用 setf( )本身的概念。换句话说，在 C++ 中没有全局格式状态的概念。每个流分别维护它自己的格式状态信息。

第二种形式的 setf( )仅影响在 flags2 中设置的标记。相应的标记首先被重置，然后依据 flags

所指定的标记设置。即使 flags 包含其他的设置标记，仅仅由 flag2 指定的那些会受到影响。两种形式的 setf( )都返回与流相关联的以前设置的格式标记。

相关函数是 unsetf( )和 flags( )。

32.6.25 setstate

```
#include <iostream>
void setstate(iostate flags) const;
```

setstate( )函数是 ios 的一个成员。

setstate( )函数设置 flags 所描述的相关联流的状态。更多细节参见 rdstate( )。

相关函数是 clear( )和 rdstate( )。

32.6.26 str

```
#include <sstream>
string str() const;
void str(string &s)
```

str( )函数是 stringstream, istringstream 和 ostringstream 的一个成员。

str( )函数的第一种形式返回一个 string 对象，该对象包含基于字符串流的当前内容。

第二种形式释放当前包含于字符串流中的字符串并用 s 所指的字符串代替。

相关函数是 get( )和 put( )。

32.6.27 stringstream, istringstream 和 ostringstream

```
#include <sstream>
explicit stringstream(ios::openmode mode = ios::in | ios::out);
explicit stringstream(const string &str,
                      ios::openmode mode = ios::in | ios::out);
explicit istringstream(ios::openmode mode=ios::in);
explicit istringstream(const string str, ios::openmode mode=ios::in);
explicit ostringstream(ios::openmode mode=ios::out);
explicit ostringstream(const string str, ios::openmode
                       mode=ios::out);
```

stringstream( ), istringstream( )和 ostringstream( )函数分别是 stringstream, istringstream 和 ostringstream 类的构造函数。这些构造与字符串相连的流。

仅指定 openmode 参数的 stringstream( ), istringstream( )和 ostringstream( )版本创建空流。带有一个 string 参数的版本初始化字符串流。

下面是演示字符串流使用情况的一个例子。

```
// Demonstrate string streams.
#include <iostream>
#include <sstream>
using namespace std;
int main()
{
  stringstream s("This is initial string.");

  // get string
```

```
  string str = s.str();
  cout << str << endl;

  // output to string stream
  s << "Numbers: " << 10 << " " << 123.2;

  int i;
  double d;
  s >> str >> i >> d;
  cout << str << " " << i << " " << d;

  return 0;
}
```

这个程序生成的输出如下所示：

```
This is initial string.
Numbers: 10 123.2
```

相关函数是str( )。

32.6.28 sync_with_stdio

```
#include <iostream>
bool sync_with_stdio(bool sync = true );
```

sync_with_stdio( )函数是ios (从ios_base继承而来)的一个成员。

调用sync_with_stdio( )会允许标准的类C I/O系统和C++基于类的I/O系统同时、安全地使用。要关掉stdio的同步功能，把false传递给sync_with_stdio( )。返回以前的设置：同步为true，非同步为false。默认时，标准流被同步。仅在先于任何其他I/O操作被调用时，这个函数才是可靠的。

32.6.29 tellg和tellp

```
#include <iostream>
pos_type tellg();
pos_type tellp();
```

tellg( )函数是istream的一个成员，tellp( )是ostream的一个成员。

C++ I/O系统管理两个与一个文件相关联的指针。一个是获取指针（get pointer），它指定下一个输入操作出现在文件的什么地方。另一个是放置指针（put pointer），它指定下一个输出操作出现在文件的什么地方。每次发生输入或输出操作时，自动按序给相应的指针加1。可以使用tellg( )决定get指针的当前位置，使用tellp( )决定put指针当前的位置。

pos_type是一种类型，它可以容纳两个函数返回的最大值。

由tellg( )和tellp( )返回的值可分别用做seekg( )和seekp( )的参数。

相关函数是seekg( )和seekp( )。

32.6.30 unsetf

```
#include <iostream>
void unsetf(fmtflags flags);
```

unsetf( )函数是ios (从ios_base继承而来)的一个成员。

使用unsetf( )函数来清除一个或多个格式标记。由flags指定的标记被清除（所有其他标记不受影响）。

相关函数是setf( )和flags( )。

32.6.31 width

```
#include <iostream>
streamsize width() const;
streamsize width(streamsize w);
```

width( )函数是ios（从ios_base中继承来的）的一个成员。

要获得当前域的宽度，使用第一种形式的width( )。它返回当前域的宽度。要设置域的宽度，使用第二种形式。这里，w变成了域宽，以前的域宽被返回。

相关函数是precision( )和fill( )。

32.6.32 write

```
#include <iostream>
ostream &write(const char *buf, streamsize num);
```

write( )函数是ostream的一个成员。

write( )函数把num个字节从buf所指的缓存中写到关联的输出流中。它返回到这个流的引用。

相关的函数是read( )和put( )。