为什么我认为流操作符不是在iomanip中定义的

今天老师在每日一问的答案中提到，要想使用流操作符需要引入iomanip头文件。我晚上仔细思考后认为可能还有其他解释。

什么是流，什么是流操作符

在弄清楚流操作符(stream operator)之前，先确认在C++中什么是流(stream)。

cpp官方文档对流(stream)给出如下定义，详见（https://cplusplus.com/reference/iolibrary/）

> A stream is an abstraction that represents a device on which input and output operations are performed. A stream can basically be represented as a source or destination of characters of indefinite length.
>
> Streams are generally associated to a physical source or destination of characters, like a disk file, the keyboard, or the console, so the characters gotten or written to/from our abstraction called stream are physically input/output to the physical device. For example, file streams are C++ objects to manipulate and interact with files; Once a file stream is used to open a file, any input or output operation performed on that stream is physically reflected in the file.

这里说，流本质是一种抽象的东西，输入和输出操作作用于流上。它在实体上可以理解为字符的最终抵达的地方。

这太抽象了，Microsoft的cpp文档给出了更直接的解释，详见（https://docs.microsoft.com/en-us/cpp/standard-library/what-a-stream-is?view=msvc-170）

> ## What a Stream Is
>
> Like C, C++ doesn't have built-in input/output capability. All C++ compilers, however, come bundled with a systematic, object-oriented I/O package, known as the `iostream` classes. The stream is the central concept of the `iostream` classes. You can think of a stream object as a smart file that acts as a source and destination for bytes. **A stream's characteristics are determined by its class and by customized insertion and extraction operators.**
>
> Through device drivers, the disk operating system deals with the keyboard, screen, printer, and communication ports as extended files. The `iostream` classes interact with these extended files. Built-in classes support reading from and writing to memory with syntax identical to that for disk I/O, which makes it easy to derive stream classes.

注意到这句话：

A stream's characteristics are determined by its class and by customized insertion and extraction operators.

它告诉我们stream在cpp中可以被当成一种对象去处理，并且由它所在的类(class)和用户自定的插入提取操作符(insertion and extraction operators)来决定。

那么，问题中的流操作符，应该就是文档中提到的 **insertion and extraction operators**

论坛geeksforgeeks随手一搜就很容易找到答案（https://www.geeksforgeeks.org/overloading-stream-insertion-operators-c/）

> In C++, stream insertion operator “<<” is used for output and extraction operator “>>” is used for input.

继续查阅csdn确认<<与>>符号就是我们所要找的“流操作符”（https://blog.csdn.net/m0_37655357/article/details/89500782）

> 2、重载流操作符<<和>>
> 当对象的数据成员是私有数据类型时，需要显示的调用函数成员才能访问和改变它的值。这样的话，输入输出对象并不方便。
>
> 幸好在C++中，通过重载流插入符<<和流提取符>>可以解决这类问题，像下面直接输出和改变对象distance的信息。
>
> ```
> cout<<distance; //distance是一个对象
> cin>>distance;
> ```

没错，<< 与>> 就是我们要找的流操作符。

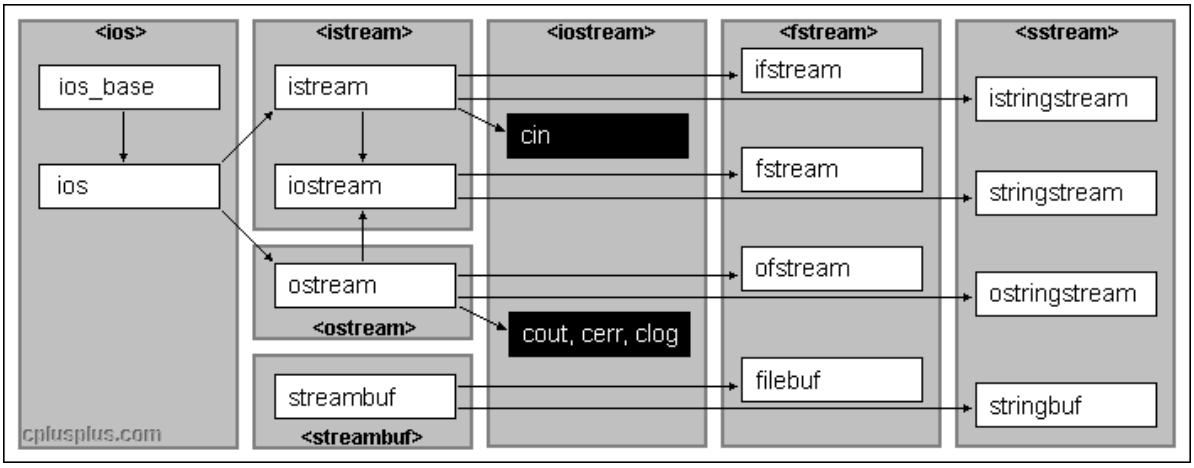
IO library

我们第一次使用流操作符是在程序HelloWord中

```
#include <iostream>
...
cout << "Hello World";
```

这个时候我们之引入了iostream包就已经可以使用 << 操作符去控制流了。所以对于流操作符的定义要么是在iostream中，要么是在iostream所包含的子包中。

查阅一下cpp官方文档关于Input/Output library的层次图：



可以看到流最开始定义在istream, ostream，我们常用的 cin, cout在iostream中得到定义。所以不出意外流操作符就应该在这三个包某个或某几个之内。

可是为什么IO library层次中没有出现iomanip的影子呢？带着疑问我接着查阅了cpp iomanip的官方文档（https://cplusplus.com/reference/iomanip/）

header

<iomanip>

IO Manipulators

Header providing parametric manipulators:

fx Parametric manipulators

setiosflags	Set format flags (function)
resetiosflags	Reset format flags (function)
setbase	Set basefield flag (function)
setfill	Set fill character (function)
setprecision	Set decimal precision (function)
setw	Set field width (function)
get_money	Get monetary value (function)
put_money	Put monetary value (function)
get_time	Get date and time (function)
put_time	Put date and time (function)

Notice that non-parametric manipulators are declared directly in <ios>.

首先看C++官方文档对于iomanip头文件的解释：

官方很明确说了，iomanip头文件提供了参数操作器(parametric manipulators)，并列出了所有的manipulators。

什么是参数操作器(parametric manipulators)呢？(https://edukedar.com/manipulators-in-c-plus)

> ## What are Manipulators in C++?
>
> manipulators are simply an instruction to the output stream that modify the output in various ways. In other words, we can say that Manipulators are operators that are used to format the data display.

原来就是控制流格式化输出的东西。

> ## Types of Manipulators in C++
>
> They are of two types one taking arguments and the other without argument.
>
> **1. Non-argument manipulators** (Without Parameters)
>
> Non-argument manipulators are also known as “Parameterized manipulators”. These manipulators require iomanip header. Examples are setprecision, setw and setfill.
>
> **2. Argumented manipulators** (With parameters)
>
> Argument manipulators are also known as “Non parameterized manipulators”. These manipulators require iostream header. Examples are endl, fixed, showpoint, left and flush.

我们熟悉的setfill, setw等都属于Parameter Manipulators他们需要引入iomanip后才能使用；而 endl, fixed等属于Non-parameter manipulators.这些Non-parameter manipulators.在最底层的ios中被直接定义好了。

可见iomanip包应该是作为IO library的补充存在的，所以官方并没有把它放到IO library中去。所以在这里就几乎可以确定，作为IO library中至关重要的流操作符，应该不会被定义于iomanip中。那流操作符到底会在哪里呢?

到底在哪

回到刚才csdn的那篇博客：

> 2、重载流操作符<<和>>
> 当对象的数据成员是私有数据类型时，需要显示的调用函数成员才能访问和改变它的值。这样的话，输入输出对象并不方便。
>
> 幸好在C++中，通过重载流插入符<<和流提取符>>可以解决这类问题，像下面直接输出和改变对象distance的信息。
>
> ```
> cout<<distance; //distance是一个对象
> cin>>distance;
> ```
>
> **实际上，这两个符号都是c++预先定义在ostream和istream类中的函数，cout和cin分别是ostream和istream类的对象。**

博客指出这两个符号是定义在ostream与istream中的函数。并且给出了自定义(重载)这两个算符的方法：

> 因此要重载这两个符号，需要在自己的类中编写重载函数，通过友元函数的形式实现函数重载。
>
> 例如下面实现<<的重载函数：
>
> (注意此处使用了ostream，因此需要在该问价的头部包含标准输入输出库：using std::ostream;
>
> 或者直接包含标准名字空间：using namespace std;)
>
> ```
> //重载流插入符<<
> friend ostream &operator<<(ostream &strm,FeetInches &obj)
> {
>     strm<<obj.feet<<" feets, "<<obj.inches<<" inches.";
>     return strm;
> }
> ```
>
> 上述函数的两个形参，strm代表<<左边的ostream对象，obj代表<<右边的自定义类对象。
>
> 该函数告诉c++当遇到【cout<<自定义对象】语句的时候，调用重载后的<<函数。
>
> 函数的返回值是一个ostream对象，是为了处理级联式的输出语句：

在这里作者仅引入了ostream库，实现了流操作符<<的重定义。 不难推测，原流操作符应该会在ostream/istream或者其子库里得到定义。

现在我们的范围就缩小到了这几个库：

- ostream
- istream
- ios
- ios_base

它在这

不多废话，直接调官方文档查ostream, 在ostream 的Class ostream中找到了<<操作符：https://cplusplus.com/reference/ostream/ostream/

Public member functions

(constructor)	Construct object (public member function)
(destructor)	Destroy object (public member function)

Formatted output:

operator<<	Insert formatted output (public member function)

Unformatted output:

put	Put character (public member function)
write	Write block of data (public member function)

Positioning:

tellp	Get position in output sequence (public member function)
seekp	Set position in output sequence (public member function)

Synchronization:

flush	Flush output stream buffer (public member function)

点开查看源代码和解释：

public member function

std::ostream::operator<<

<ostream> <iostream>

C++98 C++11

```
                    ostream& operator<< (bool val);
                    ostream& operator<< (short val);
                    ostream& operator<< (unsigned short val);
                    ostream& operator<< (int val);
                    ostream& operator<< (unsigned int val);
                    ostream& operator<< (long val);
arithmetic types (1) ostream& operator<< (unsigned long val);
                    ostream& operator<< (long long val);
                    ostream& operator<< (unsigned long long val);
                    ostream& operator<< (float val);
                    ostream& operator<< (double val);
                    ostream& operator<< (long double val);
                    ostream& operator<< (void* val);
stream buffers (2)  ostream& operator<< (streambuf* sb );
                    ostream& operator<< (ostream& (*pf)(ostream&));
manipulators (3)    ostream& operator<< (ios& (*pf)(ios&));
                    ostream& operator<< (ios_base& (*pf)(ios_base&));
```

> Insert formatted output
>
> This operator (`<<`) applied to an output stream is known as *insertion operator*. It is overloaded as a member function for:
>
> - (1) arithmetic types
>
> Generates a sequence of characters with the representation of val, properly formatted according to the locale and other formatting settings selected in the stream, and inserts them into the output stream. Internally, the function accesses the output sequence by first constructing a sentry object. Then (if good), it calls num_put::put (using the stream's *selected locale*) to perform both the formatting and the insertion operations, adjusting the stream's *internal state flags* accordingly. Finally, it destroys the sentry object before returning.
> - (2) stream buffers

> Retrieves as many characters as possible from the input sequence controlled by the *stream buffer* object pointed by sb (if any) and inserts them into the stream, until either the input sequence is exhausted or the function fails to insert into the stream. Internally, the function accesses the output sequence by first constructing a sentry object. Then (if good), it inserts characters into its associated *stream buffer* object as if calling its member function sputc, and finally destroys the sentry object before returning.
>
> - (3) manipulators
>
>   Calls `pf(*this)`, where pf may be a *manipulator*. **Manipulators are functions specifically designed to be called when used with this operator.** This operation has no effect on the output sequence and inserts no characters (unless the manipulator itself does, like endl or ends do).

注意这句话：

> **Manipulators are functions specifically designed to be called when used with this operator.**

在使用<<操作符时manipulators会被调用，这也侧面应证了上文提到的iomanip提供的多种manipulators是用来辅助流运算符处理输入的。

同样的方法，在istream中找到了>>操作符：https://cplusplus.com/reference/istream/istream/operator-free/

public member function

std::operator>> (istream) <istream> <iostream>

C++98 C++11

```
                    istream& operator>> (istream& is, char& c);
single character (1) istream& operator>> (istream& is, signed char& c);
                    istream& operator>> (istream& is, unsigned char& c);
                    istream& operator>> (istream& is, char* s);
character sequence (2) istream& operator>> (istream& is, signed char* s);
                    istream& operator>> (istream& is, unsigned char* s);
                    template<class charT, class traits, class T>
rvalue extraction (3) basic_istream<charT,traits>&
                      operator>> (basic_istream<charT,traits>&& is, T& val);
```

> Extract characters
>
> This operator (`>>`) applied to an input stream is known as *extraction operator*, and performs *formatted input*:
>
> - (1) single character
>
>   Extracts the next character from is and stores it as the value of c.
>
> - (2) character sequence
>
>   Extracts characters from is and stores them in s as a c-string, stopping as soon as either a *whitespace character* is encountered or `(width()-1)` characters have been extracted (if width is not zero). A *null character* (`charT()`) is automatically appended to the written sequence. The function then resets width to zero.
>
> - (3) rvalue extraction
>
>   Allows extracting from rvalue istream objects, with the same effect as from lvalues: It effectively calls: `is>>val`.

> Internally, the function accesses the input sequence of is by first constructing a sentry with noskipws set to `false`: this may flush its *tied stream* and/or discard leading whitespaces (see istream::sentry). Then (if good), it extracts characters from is's associated *stream buffer* object (as if calling its member functions sbumpc or sgetc), and finally destroys the sentry object before returning.
>
> Notice that if this function extracts the last character of a stream when extracting a single character *(1)*, it does not set its eofbit flag, but attempting to extract beyond it does.
>
> Calling this function does not alter the value returned by gcount on is.

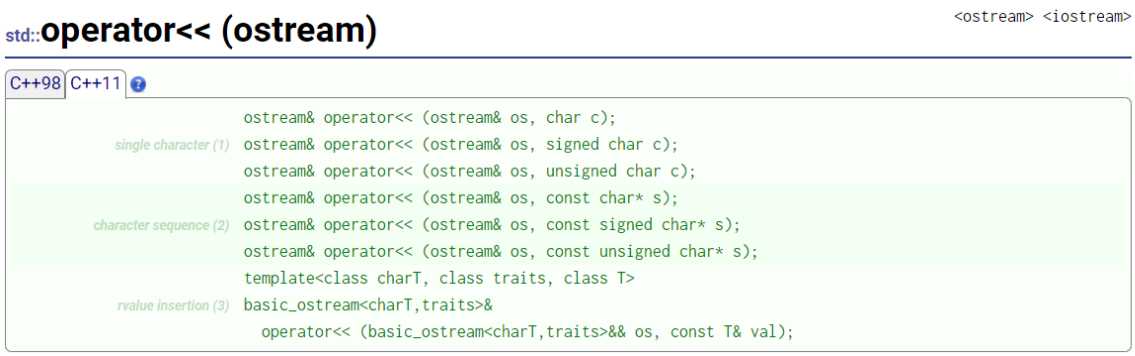
但是在ios以及ios_base中并没有发现任何<<或者>>的痕迹。https://cplusplus.com/reference/ios/ios/

只有这些吗

在ostream文档下找到这样一行小字

See operator<< for additional overloads (as non-member functions) of this operator.

很明晰，<<运算符在Cpp语言中不止一次被重载(overload)过。点开看一下详细说明：https://cplusplus.com/reference/ostream/ostream/operator-free/



可以看到所有的<<重载都和ostream(base_ostream)有关。对于 >>操作符也应该也是这个道理。

总结

- 在cpp中流(stream)是一种对象，它和输入输出密切相关，由它所属的类和用户自定的insertion and extraction operators等决定。
- 流操作符可以细分为插入操作符(insertion operator <<)和提取操作符(extraction operator>>)两种
- iomanip库提供了一系列格式化输入的措施(manipulator)，应该作为IO Library的补充存在
- 流操作符所能找到最底层的定义在ostream和istream中，所有基于他们的库被导入后都应该可以使用流操作符。