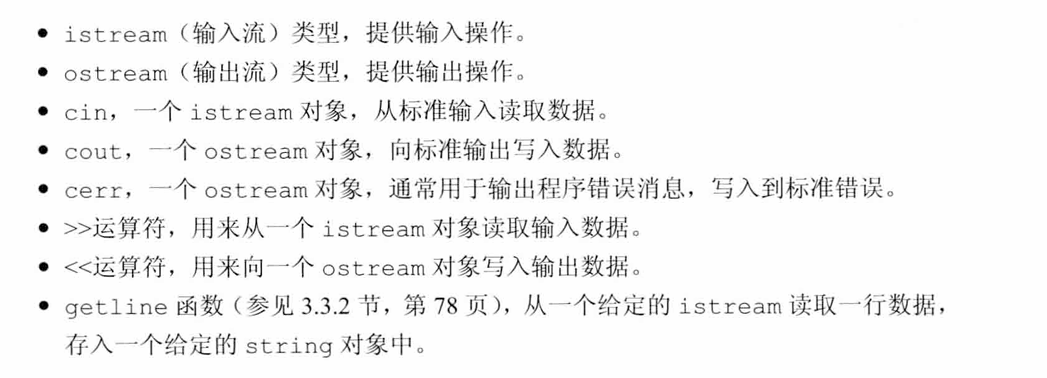
IO库提供的对象、方法、类：



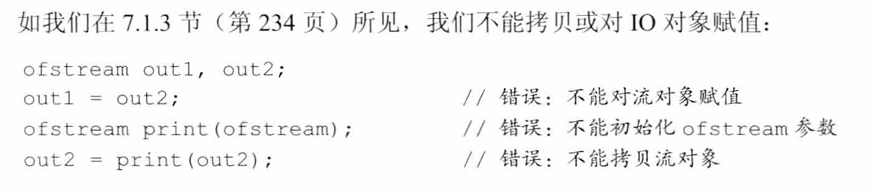
IO库提供的类和所在的头文件：



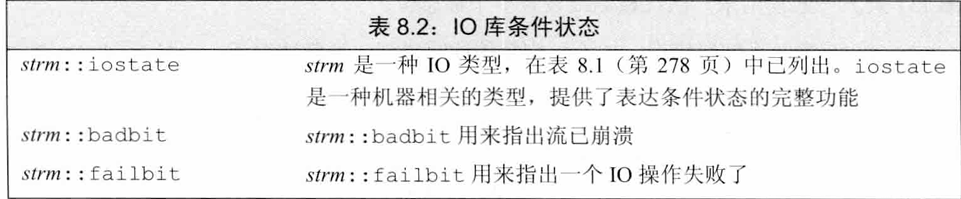
上图中，带 w 前缀的类型用来操作宽字符语言(wchar\_t)。宽字符版本的类型和函数前都有一个 w，如 wcin, wcout, wcerr。fstream 和 sstream 中的类型都继承自 iostream 中的类型。

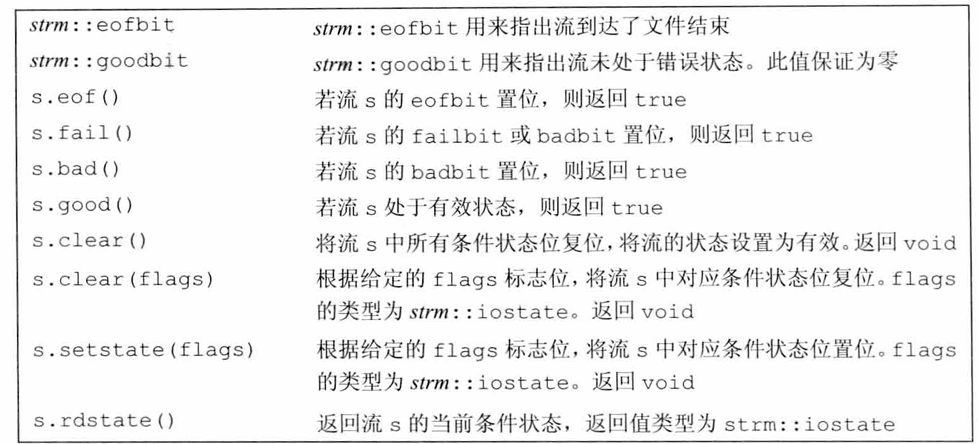
能通过>>符号读取IO库支持的任意一种流，并且也不用管存储容器的类型。这归功于运算符重载和模板编程技术，并且通过继承机制，可以把子类对象转换为父类对象来使用。

无法对IO类进行赋值、初始化、拷贝操作。进行IO操作的函数以引用方式传递和返回流，并且读写会改变流的状态，因此传递和返回的引用不能是const。

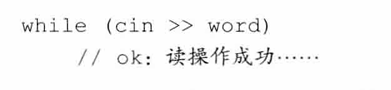


IO类提供的函数和标识，通过这些可以访问和操纵流的条件状态：





一个流一旦发生错误，后续的所有IO操作都会失效。只有一个流处于无错状态时才可以读取数据，我们可以通过下面的方式在读取数据前检查流是否有错



查询流的状态：

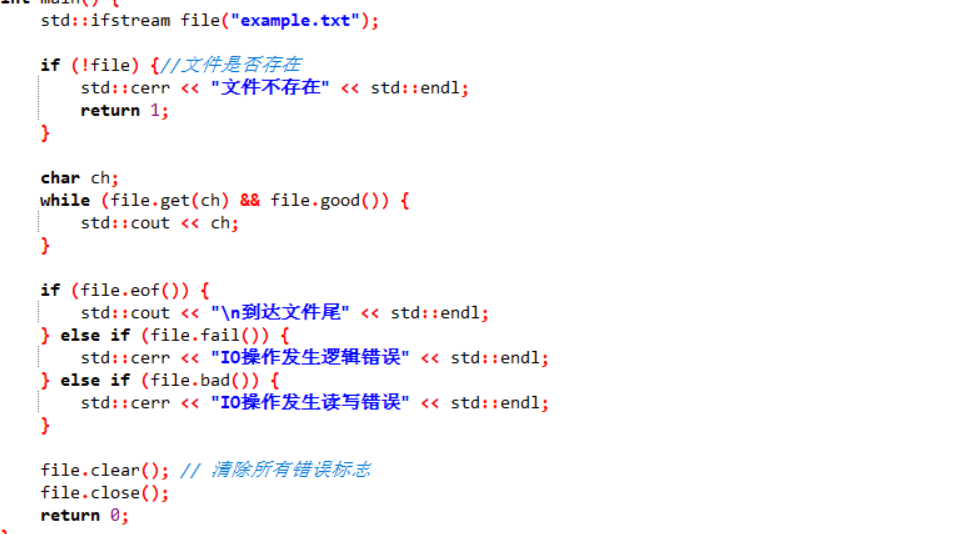
file.good()流状态是否良好，即是否没有错误。

file.bad()检查是否发生了读取或写入错误（例如硬件故障）

file.eof():检查是否已到达文件末尾

file.fail()检查是否发生了逻辑错误（例如输入类型不匹配）

file.clear()清除所有错误标志，使流恢复到良好状态

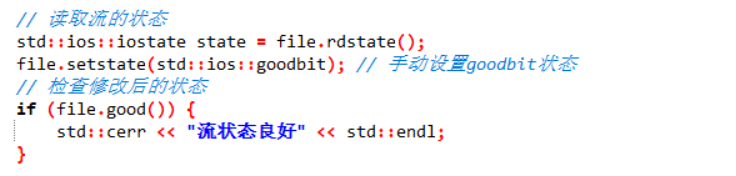


管理流的状态：

通过clear()的一个有参版本，传入使用rdstate()成员函数得到的iostate值，进行位操作修改流的状态

rdstate()读取流的状态

setstate()设置流的状态，参数会和当前状态进行按位与操作



管理输入缓冲：

每个输出流都有一个缓冲区，用来保存程序的读写数据。缓冲区里的数据可能立即打印，也可能等待缓冲区刷新一起打印。这样能把单个读写操作合成系统级的写操作，因为写操作消耗性能，使用缓冲区可以提升性能。

使得缓冲区刷新，让数据从缓冲区写入到输出设备的原因：

1.程序正常结束，比如执行到了 return。

2.缓冲区满了

3.使用操纵符，比如endl、ends、flush来显示刷新缓冲区

4.当读写被关联的流时，如读 cin 或写 cerr 都会刷新 cout 的缓冲区

5.使用操纵符 unitbuf 设置流的内部状态来清空缓冲区。

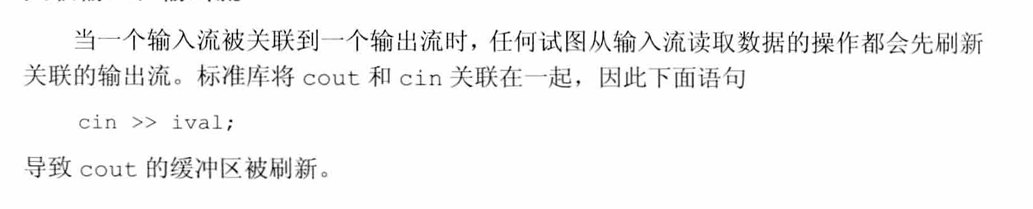
ps：endl操作符用于在输出流中插入一个换行符，并刷新流的缓冲区；ends操作符用于在输出流中插入一个空字符（'\0'），空字符标识流的结束。

std::unibuf 和 std::nounibuf 的作用

作用于输出流（”<<“左边必须是输出流），开启后所有输出操作都会使得从缓冲区立即flush到输出设备

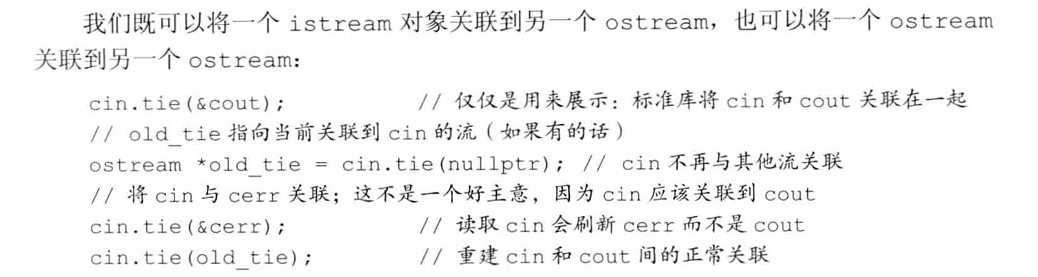
image.png

输入流和输出流被关联在一起，每次输入的时候，缓冲区会被刷新一次。并且每次从该输入流读取数据前都会先刷新缓冲区再向输入流写入。



手动将输入流和输出流关联在一起：

ps：标准库将 cin 和 cout 关联在一起。



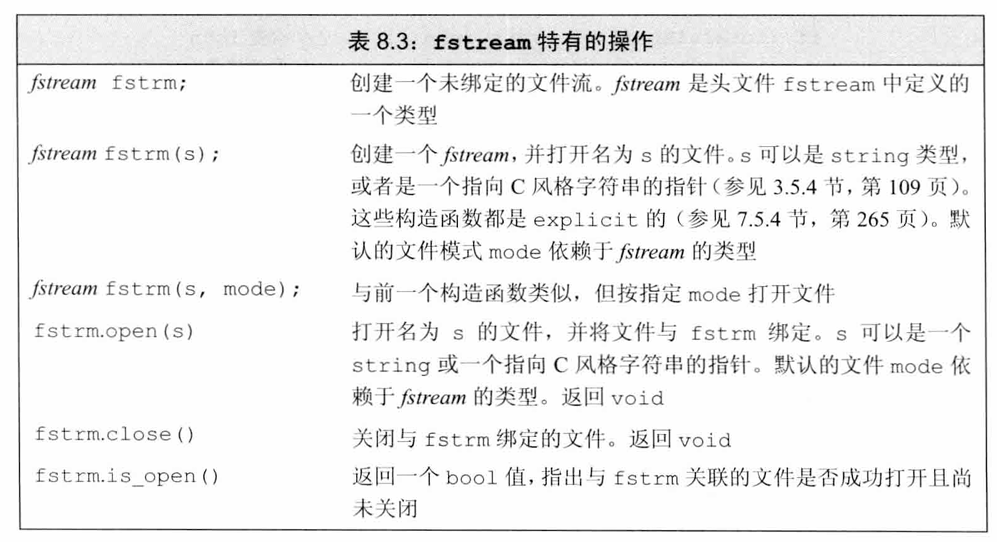
一个输入流只能关联一个输出流，而一个输出流可以被多个输入流同时关联。

文件的输入输出

头文件 fstream 中定义的 ifstream, ofstream, fstream 类型用来对文件进行读写。istream从一个文件从，ostream向一个文件写入，fstream读写都可以。

当要读写一个文件时，创建一个文件流对象并将之绑定到该文件。

fstream定义的特有操作，可以对fstream、ifstream、ofstream调用这些操作：

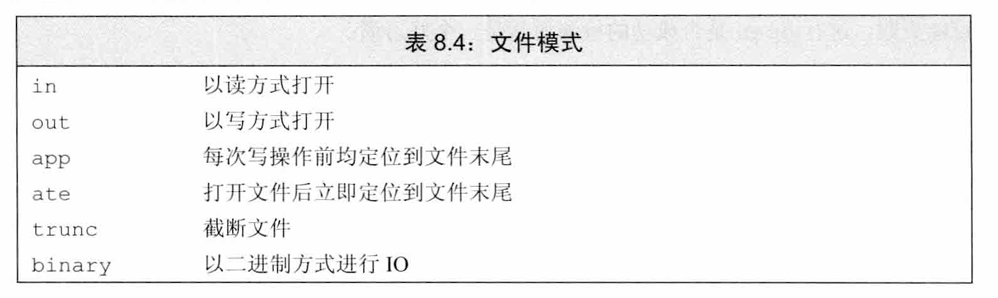


当定义了一个空的文件流对象，使用 open 函数将其与文件关联并打开文件。如果 open 失败，failebit 会被置位，建议每次 open 后检测 open 是否成功。不能对已打开的文件流调用 open。当文件关闭后，可以将文件流关联到另一个文件。当一个 fstream 对象被销毁时，close 函数会自动被调用。

ps：使用 iostream 类型的引用作为函数参数的地方，都可以使用 fstream 来代替。（因为子类拥有父类的所有成员）

文件模式：

每个流都有一个关联的文件模式，用来指出如何使用文件。

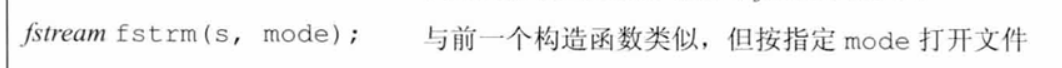


ps：

1.trunc的意思是清空当前文件中的内容，然后打开文件

2.app和ate的区别：std::ios::ate打开文件时文件指针定位在文件末尾，但可以在文件任意位置进行读写操作；std::ios::app文件指针始终在文件末尾，所有写操作都被追加到文件末尾。

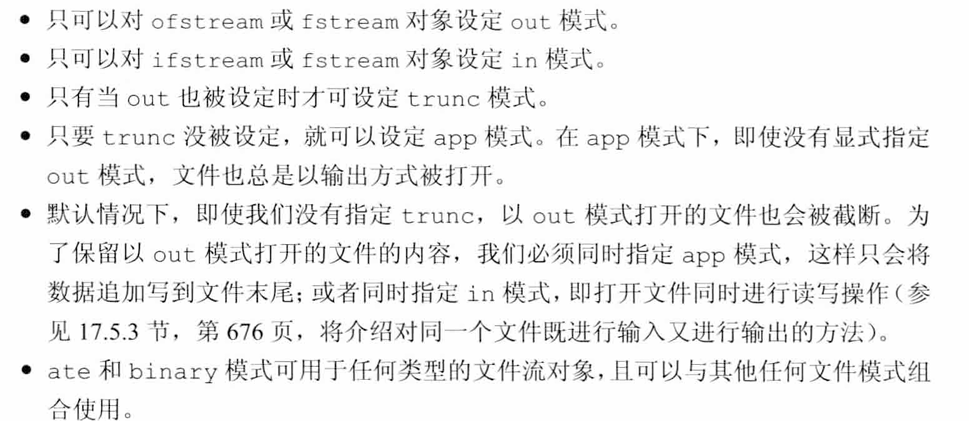
无论使用以上哪种方式打开文件，都可以指定文件模式，



fstream构造函数fstrm(s, mode)中的mode就是指定文件模式的。

指定文件模式时的限制：

强调，即使没有指定trunc，用out打开也会截断文件，除非同时指定app模式



IO库类型对象默认的文件模式：

1.std::ifstream（输入文件流）对象用于从文件读取数据。默认情况使用std::ios::in文件模式。

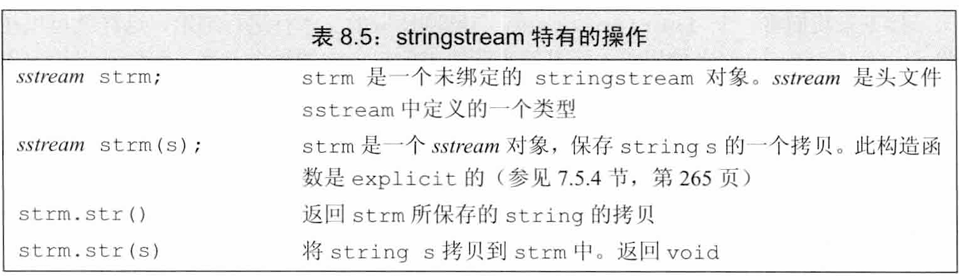
2.std::ofstream（输出文件流）对象用于向文件写入数据。默认情况使用std::ios::out文件模式。如果文件已存在，已有内容将会被截断（清空）。

3.std::fstream（文件流）对象可以同时用于读取和写入文件。默认情况使用std::ios::in | std::ios::out（读写）文件模式。如果文件已存在，默认情况下将不会截断文件内容。

string流

sstream 定义了 istringstream, ostringstream, stringstream 来读写 string。

stringstream特有操作（只能对stringstream使用，不可以对其他IO类使用）：



istringstream读取输入字符串例子：



ostringstream输出到string的例子：

