# 1.IO类：

使用过的IO类型和对象都是操纵char数据的，这些对象都是关联到用户的控制台窗口的。

|  |  |
| --- | --- |
| 头文件 | 类型 |
| iostream  （控制台） | istream，wistream：从流中读取  ostream，wostream：写到流中去  iostream，wiostream：对流进行读写 |
| fstream  （文件） | ifstream，wifstream：从文件中读取  ofstream，wofstream：写到文件中去  fstream，wfstream：读写文件 |
| sstream  （string对象） | istringstream，wistringstream：从 string 对象中读取 ostringstream，wostringstream：写到 string 对象中去 stringstream，wstringstream：对 string 对象进行读写 |

PS:带“w”的是宽字符类型。为了支持宽字符语言，标准库定义了一组类型和对象来操纵wchar\_t类型的数据。宽字符版本的类型和函数的名字以一个w开始，例如：wcin、wcout。

标准流特性都可以无差别地应用于普通流、文件流和string流，以及char或宽字符流版本。这是通过继承机制实现的，比如ifstream和istringstream就继承自istream。

## 1.1IO对象无拷贝或赋值：

由于不能拷贝或赋值，因此我们不能将形参或返回类型设置为流类型，读写一个IO对象会改变其状态，因此传递和返回的引用不能是const。

## 1.2条件状态：

IO类定义了一些函数和标志，可以帮助我们访问和操纵流的条件状态。

|  |  |
| --- | --- |
| strm::iostate | 机器相关的整型名，由各个 iostream 类定义，用于定义条件状态，提供了各表达条件状态的完整功能 |
| strm::badbit | strm::iostate 类型的值，用于指出被破坏的流，表示系统级错误时被置位，是不可恢复错误，一旦badbit被置位，流无法使用 |
| strm::failbit | strm::iostate 类型的值，用于指出失败的 IO 操作，用于表示可恢复错误（如期望读取数值却读出一个字符）或文件结束位置时被置位，流还可以使用，可以修正 |
| strm::eofbit | strm::iostate 类型的值，用于指出流已经到达文件结束符，用于表示文件结束位置被置位。 |
| strm::goodbit | strm::goodbit类型的值，用于指出流处于错误状态，此值保证为0 |
| s.eof() | 如果流 s 的 eofbit 值被置位，则该函数返回 true |
| s.fail() | 如果流 s 的 failbit 或badbit值被置位，则该函数返回 true |
| s.bad() | 如果流 s 的 badbit 值被置位，，则该函数返回 true |
| s.good() | 如果流 s 所有错误位均未置位，则该函数返回 true |
| s.clear() | 将流 s 中的所有状态位复位，将流的状态设置为有效，返回void |
| s.clear(flag) | 根据给定的flags标志位，将流s中对应条件状态位复位。flag 的类型是strm::iostate，返回void |
| s.setstate(flag) | 根据给定的flags标志位，将流s中对应条件状态位复位。flag 的类型是strm::iostate，返回void |
| s.rdstate() | 返回流 s 的当前状态，返回值类型为 strm::iostate |

一个流一旦发生错误，其上后续的IO操作都会失败，由于流可能出于错误状态，因此代码通常应该在使用一个流之前检查它是否处于良好状态，最简单的方法是将它当作一个条件来使用（只能检测流是否有效，不能返回发生了什么），实际上，等价于!fail()，而eof和bad操作只能表示特定的错误。比如要求输入数字，但却输入了字符串，流就会进入错误状态。

IO库定义了一个与机器无关的iostate类型，它提供了表达流状态的完整功能，这个类型作为一个位集合来使用。IO库定义了4个iostate类型的constexpr值，表示特定的位模式，这些值用来表示特定类型的IO条件，可以与位运算符一起使用来一次性检测或设置多个标志位。

如果badbit、failbit、eofbit人一个被置位，则检测流状态的条件会失败。

strm是一种IO类型。

auto old\_state = cin.rdstate();//记住cin的当前状态

cin.clear();//使cin有效

process\_input(cin);//使用cin

cin.setstate(old\_state);//将cin置为原有状态

cin.clear(cin.rdstate & ~cin.failbit & ~cin.badbit);//将failbit和badbit复位，保持eofbit不变。

## 1.3管理输出缓冲：

每个输出流都管理一个缓冲区，用来保存程序读写的数据。有了缓冲机制，操作系统可以将程序的多个输出操作组合成单一的系统级写操作。由于设备的写操作可能很费时，允许操作系统将多个输出操作组合为单一的设备写操作可以带来很大的提升。

导致缓冲刷新（数据真正）的原因：

①程序正常结束，main函数的return操作，

②缓冲区满的时候，

③使用操纵符如endl、flush、ends来显式刷新缓冲区，flush刷新缓冲区但不输出任何额外的字符；ends向缓冲区插入一个空字符，然后书信缓冲区，

cout<<”hi”<<flush;

cout<<”hi<<ends;”

④每个输出操作后，用操纵符unitbuf设置流的内部状态，来清空缓冲区,它告诉流在接下来的每次写操作之后都进行一次flush操作，而nounitbuf操纵符则重置流，使其恢复使用正常的系统管理的缓冲区刷新机制。默认情况下，对cerr是设置unitbuf的，因此写到cerr的内容都是立刻刷新的，

cout<<nounitbuf;//回到正常的缓冲方式

⑤一个输出流可能被关联到另一个流，当读写被关联的流时，关联到的流的缓冲区会被刷新，

如果程序异常终止，它所输出的数据很可能停留在输出缓冲区中等待打印。交互式系统通常应该关联输入流和输出流，这意味着所有输出，包括用户提示信息，都会在读操作之前被打印出来。文件输入

# 2.文件输入输出：

除了继承来自iostream类型的行为之外，fstream中定义的类型还增加了一些新的成员来管理与流相关的文件（当然，原来的还要继承）。

|  |  |
| --- | --- |
| fstream fstrm; | 创建一个未绑定的文件流，fstream是头文件fstream中定义的一个类型 |
| fstream fstrm(s); | 创建一个fstream，并打开名为s的文件。s可以是string类型，或者是一个指向C风格字符串的指针，这些构造函数是explicit的，默认的文件模式mode依赖于fstream类型 |
| fstream fstrm(s,mode); | 与前一个构造函数类似，但按指定mode打开文件 |
| fstrm.open(s); | 打开名为s的文件，并将文件与fstrm绑定，视情况打开微读或写模式。s可以是一个string或一个指向C风格的字符串的指针，默认的文件mode依赖于fstream类型，返回void。 |
| fstrm.close(); | 关闭与fstrm绑定的文件，返回void |
| fstrm.is\_open(); | 返回一个bool值，指出与fstrm关联的文件是否成功打开且尚未关闭 |

## 2.1使用文件流对象：

创建文件流对象时，我们可以提供文件名（可选），如果提供了文件名，则open会自动被调用。

ifstream in(file);//构造一个ifstream并打开给定文件，文件名由string类型的参数ifile指定

ofstream out;

2.1.1用fstream代替iostream&：

在要求使用及类型对象的地方，可以用继承类型的对象来替代。比如有一个函数接受一个ostream&参数，在调用函数的时候，可以给他传递一个ofstream对象。

ifstream input(argv[1]);//打开销售记录文件

ofstream output(argv[2]);//打开输出文件

output.open(ifile + “.copy”);//打开指定文件，ifile是string类型。

if(out)//检查open是否成功，如果成功，就可以使用文件了，和之前的if(cin)相同的原理

一旦一个文件流被打开，它就能保持与对应文件的关联。对于一个已经打开的文件，如果调用open失败，failbit会被置位，随后的文件流的操作都会失败；如果open成功，则使得good()为true。为了将文件流关联到另一个文件，必须首先关闭已经关联的文件，但是当一个fstream对象离开其作用域时，与之关联的文件会自动关闭。

## 2.2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，mode 是以下标志符的一个组合：

|  |  |
| --- | --- |
| ios::in | 为输入(读)而打开文件 |
| ios::out | 为输出(写)而打开文件 |
| ios::ate | 打开文件后立即定位到文件末尾 |
| ios::app | 每次写操作前均定位到文件末尾 |
| ios::trunc | 截断文件，如果文件已存在则先删除该文件 |
| ios::binary | 二进制方式进行IO |

这些标识符可以被组合使用，中间以”或”操作符(|)间隔。

①只有当out也被设定时才可设定trunc模式，

②只要trunc没被设定，就可以设定app模式，在app模式下即使没有显式指定out模式，文件也总是以输出方式被打开，

③默认情况下，即使我们没有指定trunc，以out模式打开的文件也会被默认截断，加上trunc，为了保留以out模式打开的文件的内容，我们必须同时指定app模式，这样只会将数据追加写到文件末尾，或者同时指定in模式，即打开文件同时进行读写操作。

④ate和binary模式可用于任何类型的文件流对象，且可以与其他任何文件模式组合使用。

⑤与ifstream关联的文件默认以in模式打开；与ofstream关联的文件默认以out模式打开；与fstream关联的文件默认以in和out模式打开。

⑥当我们打开一个ofstream时，文件的内容会被清空丢弃，阻止清空的方法是指定app模式。

⑦使用文件模式标识符时要在前面加上ios::

# 3.string流：

头文件sstream中定义的类型都继承自我们已经使用过的iostream头文件中定义的类型。

|  |  |
| --- | --- |
| sstream strm; | strm是一个未绑定的stringstream对象 |
| sstream strm(s); | strm是一个sstream对象，保存string s的一个拷贝，此构造函数是explicit的 |
| strm.str(); | 返回strm所保存的string的拷贝 |
| strm.str(s); | 将string s拷贝到strm中，返回void |

## 3.1使用istringstream：

struct PersonInfo

{

string name;

vector<string> phones;

}

string line,word;

vector<PersonInfo> people;

while(getline(cin,line))

{

PersonInfo info;

istringstream record(line);

record>>info.name;

while(record>>word)//可以自动类型转换、识别

info.phones.push)back(word);

people.push\_back(info);

]

## 3.2使用ostringstream:

当我们逐步构造输出，希望最后一起打印时，ostringstream是很有用的。

for(const auto &entry:people)

{

ostringstream formatted,badNums;

for(const auto &nums:entry.phones)

{

if(!valid(nums)) badNums<<” ”<<nums;//valid完成电话号码验证

else formatted<<” ”<<format(nums);//format将字符串格式化

}

if(badNums.str().empty())

os<<entry.name<<” ”<<formatted.str()<<endl;

else cerr<<”input error:”<<entry.name<<” invalid number(s) ”<<badNums.str()<<endl;

}