第八章 IO库

> C语言不直接处理输入输出，而是通过一组定义在标准库中的**类型**来处理IO。
>
> 这些类型支持从设备读取数据、向设备写入数据的IO操作

IO类

IO库类型和头文件

头文件	类型
iostream	istream——从流中读取数据 ostream——向流中写入数据 iostream——读写流
fstream	ifstream——从文件中读取数据 ofstream——向文件中写入数据 fstream——读写文件
sstream	istringstream——从string中读取数据 ostringstream——向string中写入数据 stringstream——读写string

- 为了支持宽字符的语言，标准库定义类一组类型和对象来操作`wchar_t`类型的数据；
  上面所有类型前加上`w`即是操作宽字符对象的类
- 概念上，设备的类型和字符的大小都不会影响我们要执行的IO操作；类似的，我们也不用管读取的字符需要哪种字符类型来存储

  标准库通过继承机制来使我们忽略不同类型的流的差异

  > `ifsream`和`istringstream`**都继承自**`istream`，所以对`istream`的操作都可以用到这两个类上

IO对象

- 不能拷贝或对IO对象赋值
  - IO对象作函数形参或返回类型通常用引用形式

> 也可以用指针类型

- 读写一个IO对象会改变其状态
  - 传递和返回的引用是不能为 `const`

条件状态

- IO库条件状态：帮助我们访问和操作流的条件状态

状态	含义
iostate	一种机器相关的类型，提供类表达条件状态的完整功能。
badbit	指出流已崩溃 发生系统级错误，无法恢复，流无法再使用
failbit	指出一个IO操作失败 发生可恢复错误时（包括到达文件末尾位置），failbit被置位
eofbit	流到达了文件结束 到达文件末尾结束位置，
goodbit	指出流未处于错误状态，此值保证为0

> - iosate类型作为一个**位集合**使用，对应标志位位为1则说明出现了该错误，全0就是goodbit状态
> - 可以与与运算一起使用来一次性检查或设置多个标志位
> - 后续的四个是IO库内定义的4个iosate类型的constexpr的值，这些值用用来表示特定类型的IO条件
>
>   每个IO类中都有这些数据成员

```
//ios_base.h库内显示
enum _Ios_Iostate
{
    _S_goodbit      = 0,
    _S_badbit       = 1L << 0,
    _S_eofbit       = 1L << 1,
    _S_failbit      = 1L << 2,
    _S_ios_iostate_end = 1L << 16,
    _S_ios_iostate_max = __INT_MAX__,
    _S_ios_iostate_min = ~__INT_MAX__
};
```

  - 注意，一个流一旦发生错误，其上后续的IO操作都会失败

- 相关成员函数

函数	含义
s.eof()	判断流是否处于eofbit状态
s.fail()	判断流是否处于failbit或badbit状态
s.bad()	判断流是否处于badbit状态
s.good()	若流处于有效状态，返回true
s.clear()	将流s中所有状态位复位，将流的状态位设置为有效，返回`void`
s.clear(flags)	根据给定的flags标志位，将流s对应的状态位复位。 flags的类型是`strm::iostate`。返回void
s.setstate(flags)	根据给定的flags标志位，设置流s对应的状态位。 flags的类型是`strm::iostate`。返回void
s.rdstate()	返回流当前的条件状态，返回值类型是`strm::iostate`

> strm代表对应的流类型
>
> 可以直接通过这些函数来判断流的状态（当然，实际上还是通过与运算来判断各位的情况，只是封装起来了）

- 管理条件状态：

  流对象的`rdstate`成员返回一个iostate值，对应流的当前状态

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

```
os<<"please enter a value:";        // 文件串可能立即打印出来，也可能被操作系统保存在缓冲区中稍后打印
```

- 导致缓冲刷新（即，**数据真正写到输出设备或文件**）的原因
  - 程序正常结束
  - 缓冲区满，需要刷新后，新数据才能继续写到缓冲区
  - 操作符来显式的刷新缓冲区
    - `endl`：换行并刷新
    - `flush`：只刷新缓冲区
    - `ends`：向缓冲区插入一个空字符，然后刷新

- 在每个输出操作之后，可以用操作符 `unitbuf` 设置流的内部状态，来清空缓冲区

```
//
cout<<unitbuf;        // 之后的每次写操作之后都进行一次flush操作
cout<<nounitbuf;      // 重置流，恢复默认
```

> 默认情况下， `cerr` [1] 是设置 `unitbuf` 的

- 一个输出流可能被关联到另一个输出流。

  在这种情况下，当读写被关联的流时，关联到的流的缓冲区会被刷新

```
/*tie()—不带参数版本
如果该对象当前关联到一个输出流，则返回的就是指向这个流的指针；
如果对象没有关联到输出流，则返回空指针
*/
    auto p=cin.tie();
    *p<<"a";
/*tie(&o)-带参数版本
接受一个指向ostream的指针，将自己关联到此ostream
cin.tie(&cout)
*/
```

> 默认情况下， `cin` 和 `cerr` 都被关联到 `cout` 。因此读 `cin` 或写 `cerr` 都会导致之前的 `cout` 的缓冲区被刷新

> 如果程序崩溃，输出缓冲区不会被刷新

文件输入输出

fstream特有操作

操作	含义
`fstream fstrm`	创建一个未绑定的文件流
`fstream fstrm(s)`	创建一个绑定的文件流 s可以是一个string，或者是一个指向C风格字符串的指针，或者是一个字面值
`fstream fstrm(s,mode)`	
`fstream.open(s)`	重新绑定到一个文件
`fstream.close()`	关闭与fstrm绑定的文件，返回void
`fstream.is_open()`	返回一个bool值，指出与fstrm关联的文件**是否打开且尚未关闭**

> 继承类才有的操作，不能对其他IO类型调用这些操作

- 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；

  一旦文件成功关闭，我们可以打开新的文件

```
in.close();
in(ifile +  "2");
```

- 当一个 `fstream` 对象被销毁时， `close()` 会自动调用

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。

模式	含义
in	只读
out	只写（默认截断文件）
app	每次写操作前均定义到文件末尾
ate	打开文件后立即定位到文件末尾
trunc	截断文件
binary	以二进制进行IO

> 截断文件可以理解为清空文件内容，但是不删除文件

- 只可以对 `fstream` 或 `fstream` 对象设立 `out` 模式

- 只可以对· `ifstream` 或 `fstream` 对象设立 `in` 模式
- 只有当 `out` 也被设定时，才可设定 `trunc` 模式
- 只要 `trunc` 模式没被设计，就可以设定 `app` 模式

  在app模式下，即使没有显示指定out模式，文件也总是以输出方式被打开
- 默认情况下，即使没有指定 `trunc` 模式，以 `out` 模式打开的文件也会被截断。

  为了保留以 `out` 模式打开的文件内容，我们别处同时指定 `app` 模式，这样会将数据追加到文件末尾；或同时指定 `in` 模式，即打开文件同时进行读写操作
- `ate` 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

> 每个文件流类型都定义了一个默认文件模式
>
> - `ifstream` ——in模式
> - `ofstream` ——out模式
> - `fstream` ——in和out模式
>
> 实际上每次打开(`open`)一个文件都要重新指定一个文件模式，只是使用默认模式则不需要第二个参数。

String流

`sstream` 头文件定义了三个类型来支持 **内存IO** ,这些类型可以向string写入数据、从string中读取数据，就像string是一个IO流一样

stringstream特有操作

操作	含义
sstream strm	strm是一个未绑定的stringstream对象
sstream strm(s)	strm是一个sstream对象，保存一个string s的拷贝。此构造函数是 `explicit` [2] 的
strm.str()	返回strm所保存的string拷贝
strm.str(s)	将string s拷贝到strm中。返回void

- 当我们的某些工作是对整行文本进行处理，而 **其他工作是处理行内的单个单词时** ，通常可以使用 `istringstream`

```
getline(cin, line); // shenyanyu 123 456
istringstream record(line);
record >> info.name;
while(record >> tel);
```

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

> 即先用ostringstream对象保存起来，最后在输出
>
> 为什们不用一个临时变量保存呢？----> stringstrem到底有什么用？（除了类型转换）

---

1. 标准错误输出(第1章) ↩

2. 执行拷贝初始化时，只能使用直接初始化，编译器不会在自动转换过程中使用构造函数(第七章) ↩