C++程序设计精要教程

华中科技大学

第14章　流及类库

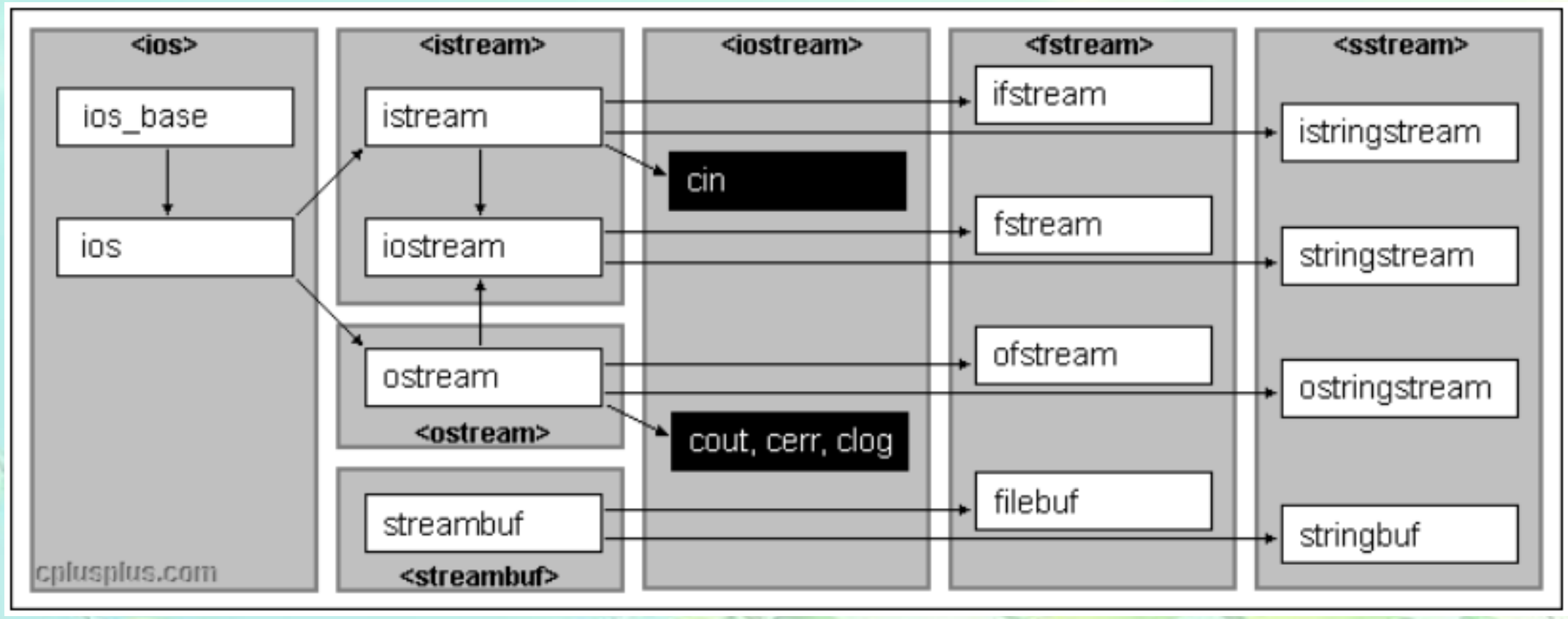
◆14.1 流类概述

- 流类实现从文件或缓冲区（字符串）中读取/存贮格式化的数据（数据从一个地方流动到另一个地方）。
- 流类包括：文件流、字符串流。
- 流操作可以直接输入/输出，也可以带缓冲的输入/输出。

第14章　流及类库

◆14.1 流类概述

流类结构图



第14章　流及类库

◆14.1 流类概述

- **ios_base、ios、streambuf 都是虚基类（不能实例化）。**
- **ios的派生类istream和ostream 实现了对流进行格式化I/O和错误处理的操作。**
- **streambuf 定义了对流缓冲区操作的接口，是流与物理设备的缓冲接口，其派生类 filebuf 和 stringbuf 被用来实现文件流和字符串流的操作。**
  - **●文件流：ifstream、ofstream、fstream**
  - **●字符串流：istringstream、ostringstream、stringstream**

第14章　流及类库

◆14.1 流类概述

- **操作系统将键盘、显示器、打印机等映射为文件。**
- **C++预定义了4个流类对象，即 cin、cout、cerr 和 clog 标准流类对象，当C++程序开始执行时，这4个流类对象已被构造好，且不能被应用程序析构**
  - **extern istream_withassign　cin;　//相应于 stdin**
  - **extern ostream_withassign　cout;　//相应于 stdout**
  - **extern ostream_withassign　cerr;　//相应于 stderr**
  - **extern ostream_withassign　clog;　//相应于有缓冲的 cerr**
- **cerr与clog之间的区别是cerr没有缓冲，发送给cerr的内容立即输出。**

第14章　流及类库

◆14.2 输出流

- 输入、输出流类定义了流类最基本的操作，包含在头文件 iostream 中。
- 输出流通过重载左移运算符 << 实现输出，其左操作数为 ostream_withassign 类型的对象cout，右操作数为所有简单类型的右值表达式

  例如：cout << "Hello!\n";
- 该语句隐含地调用 cout.operator<<(const char *str)，该函数输出参数str所指定的字符串，并返回ostream_withassign类型的引用 cout
- 上述函数调用的结果可进一步作为 << 的左操作数

第14章　流及类库

◆14.2 输出流

- 运算符<<重载后仍然保持自左至右的结合方式，因此，可以一次自左至右地输出多个右值表达式

```
cout << "i=" << i << ", d=" << d << "\n";
```

- 由于重载不改变运算符的优先级：

```
cout << "sum=" << 3+5 << "\n";
cout << “x & y=" << (x & y) << "\n";
```

第14章　流及类库

◆14.2 输出流

- **输出流为运算符<<预定义的右操作数的数据类型有：**
  char、short、int、long 等有符号或无符号的整数类型
  char *、float、double、long double 和 void * 等类型
- **所有的输出按 printf 规定的转换规则进行转换**
  例如，下面的两个输出语句产生完全一样的输出结果：

```
int   m;
long  n;
cout<<m<<'\t'<<n;
printf("%d\t%ld", m, n);
```

第14章　流及类库

- 输出格式：由cout各种状态标志确定
- 状态标志：由ios中public类型的枚举量定义

```
enum  {
      skipws,          //输入时跳过空白：空格、回车、换行及制表符等
      left ,           //左对齐输出
      right ,          //右对齐输出
      internal,        //在符号或基指示后填补
      dec,             //按十进制转换
      oct,             //按八进制转换
      hex,             //按十六进制转换
      showbase,        //在输出中使用基指示
      showpoint,       //在浮点输出中显示小数点
      uppercase,       //大写十六进制输出
      showpos,         //对正整数显示 +
      scientific,      //用科学计数法表示
      fixed,           //用小数点表示浮点数
      unitbuf,         //所有流在输出后刷新
      stdio            //在输出到stdout, stderr后刷新
};
```

将256用16进制显示

```
cout << hex << 256
```

第14章　流及类库

◆14.2 输出流

- 可以使用以下函数成员来读取、设置和清除标志：

```
long  flags( );              //读取字符格式标志，如 cout.flags();
long  flags(long);           //设置字符格式标志
long  setf(long, long);      //清除和设置字符格式标志
long  setf(long);            //设置字符格式标志
long  unsetf(long);          //清除字符格式标志
```

- 改变输出格式：可以使用操纵符改变输出宽度、填充字符等与输出格式有关的变量
  - 操纵符可以同输入／输出的变量或数据一起使用
  - 所有的操纵符都定义在 iomanip.h 中，引用前须包含 #include

第14章流及类库

【例14.1】使用操纵符改变输出格式

```
#include <iostream>
#include <iomanip>
using namespace std;
void main(void)
{
    int  i = 3456, j = 9012, k = 78;
    cout << setw(6) << i << j << k << "\n";
    cout << setw(6) << i << setw(6) << j << setw(6) << k;
    //setw(int)对输出流的影响只是暂时的
}
```

上述程序产生的输出为：

```
3456901278
3456   9012     78
```

第14章　流及类库

◆14.2 输出流

- 带参数操纵符函数有 setfill、setprecision、setiosflags、resetiosflags、setbase 等
- 程序可以定义自已的操纵符函数, 但不能带参数 .C++预定义的操纵符函数有：

```
dec;                    //设置十进制转换
hex;                    //设置十六进制转换
oct;                    //设置八进制转换
ws;                     //提取空白字符
endl;                   //插入回车并刷新输出流
ends;                   //插入空字符以终止串
setbase(int);           //设置进制标志为0,8,10,16。0表示缺省为十进制
resetiosflags(long);//清除格式位
setiosflags(long);      //设置格式位
setfill(int);           //设置填充字符
setprecision(int);      //设置浮点精度位数
setw(int);              //设置域宽
```

- 对于不带参数的dec ~ ends操纵符函数，调用时不写括号，它们对输出流的影响是长久的。

第14章流及类库

【例12.2】定义输出流的格式

```
#include <iostream>
using namespace std;
void main(void)
{
    int  i = 12;
    cout << hex << i << i;
    cout << i << i << endl;
    cout << dec << i << i;
    cout << i << i << endl;
}
```

上述程序的输出为：

```
cccc
12121212
```

第14章　流及类库

◆14.2 输出流

- C++为流定义了一些输出函数成员，这些函数是以字符或块为单位操作的。
- 当输出的数据为字符类型时，输出函数按无符号和有符号字符进行重载。

原型如下：

```
ostream &flush( );                          //刷新输出流
ostream &put(char);                         //输出一个字符
ostream &seekp(long);                       //确定输出位置
ostream &seekp(long, seek_dir);             //确定输出位置
long tellp( );                              //读取输出位置
ostream &write(const char*, int n);         //输出一个字符块
```

第14章　流及类库

◆14.3 输入流

- 从流中输入（或称提取），输入流通过重载运算符 >> 实现输入。
- 重载后运算符函数>>的左操作数为istream类型的对象，右操作数为预定义类型的引用 。
- 在缺省情况下，用运算符>>输入时将先跳过空白符，然后输入对应于输入对象的字符。
- 是否跳过空白符由ios定义的skipws确定，若清除该标志将不跳过空白符。
- 可通过操作符ws设置skipws标志，skipws被缺省设置为跳过空白符。

第14章流及类库

【例12.4】输入流的用法

```
#include <iomanip>
#include <iostream>
using namespace std;
void main(void)
{
    char c, d, s[80];
    long f;
    f = cin.flags( );              //返回格式化标志，缺省为跳过空白
    f = cin.flags(0L);             //设置格式化标志，返回原格式化标志
    cin >> c >> d;                 //不跳过空白字符输入
    cin >> ws >> c >> d;           //跳过空白字符输入
    cin.flags(f);                  //恢复原格式化标志为跳过空白
    cin.width(sizeof(s)-1);        //避免溢出
    cin >> s;                      //跳过空白输入字符串
}
```

第14章流及类库

◆ 14.4 重载输入/输出流的运算符>>和<<

对于输入/输出流的运算符 >> 和 <<，C++重载了简单的数据类型 (如 int，char * 等)，对于复杂的对象，则需在类中重载 >> 和 << 。

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
using namespace std;
class A {
public:
   int  age;
   char  name[50];
   A( ) { age = 0;  name[0] = 0; }
};
ostream &operator<<(ostream &s, A &a)
{
    return s << a.name << ", " << a.age << "\n";
}
```

```
istream &operator>>(istream &s, A &a)
{
    cout << "Name: ";  s >> a.name;
    cout << "age: " ;  s >> a.age;
    return s;
}

int main()
{
    A  a;
    cin >> a;     //如果没有重载 >> ?
    cout << a;   //如果没有重载 << ?
}
```

第14章　流及类库

◆14.5 重定向 cin 和 cout

cout和cin可以被重定向，而 cerr 不能。

cout和cin 一般表示屏幕和键盘。

下面的程序重定向cout和cin到文件：

cout输出的东西被保存到 2.txt，

cin输入的东西来自文件 1.txt。

```
#include <iostream>
using namespace std;
int main()
{
    freopen("1.txt", "r", stdin);
    freopen("2.txt", "w", stdout);
    cout << “hello\n”;   //将 hello\n 写入2.txt
    cout << 12345;       //将 12345 写入2.txt
    int x, y;
    cin >> x >> y;   //从1.txt中读取2个整数给x和y
}
```

第14章　流及类库

◆14.6 文件流

- 文件流类定义了文件I/O操作，包含在头文件 fstream 中。
- 文件流类包括：

  输入流类 ifstream，只能从文件中读取数据

  输出流类 ofstream，只能写数据到文件中

  输入输出流类 fstream（将ifstream和ofstream合到一起）
- 文件流对象必须在文件打开后才能输入／输出，在文件关闭后才能再次打开文件。定义文件流对象和打开文件可以同时进行，例如：

```
ifstream f1(“input.dat”);
ofstream f2(“output.dat");
fstream f("1.dat", ios::in | ios::out | ios::binary);
```

第14章　流及类库

◆14.6 文件流

●在缺省情况下，文件用文本模式打开，类ios定义了多种文件打开模式：

ios::app	在文件尾追加数据
ios::ate	在已打开文件上找到文件尾
ios::in	打开的文件供读
ios::out	打开的文件供写，缺省为trunc方式
ios::binary	以二进制方式打开文件
ios::trunc	若文件存在，则消除原文件内容
ios::nocreate	若要打开的文件不存在，则打开失败
ios::noreplace	除非同时设置ate或app，否则文件存在时打开失败

第14章流及类库

【例14.5】使用文件流编写文件拷贝程序。

```
#include <fstream>
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
    ifstream f1;
    ofstream f2;
    char ch;
    if (argc != 3) {
        cerr << "Parameters error!\n";
        return 1;
    }
    f1.open(argv[1], ios::in+ios::binary);
    if ( !f1 ) {        //若文件不存在
        cerr << "Source file open error!\n";
        return 1;
    }
    f2.open(argv[2], ios::out+ios::binary);
    if ( !f2 ) {
        cerr << "Object file open error!\n";
        f1.close( );
        return 1;
    }
    while ( f1.get(ch) )  f2.put(ch);
    f1.close( );
    f2.close( );
    return 0;
}
```

第14章　流及类库

◆14.7 字符串流

- 与sscanf和sprintf的功能类似，字符串流实现了从缓冲区（字符串）中提取/写入格式化的数据。
- C++定义了2种字符串流，一种在 sstream 中定义，另一种在 strstream 中定义。它们实现的功能基本一样。
- strstream 是基于char *编写的，sstream 则是基于 std::string 编写的。因此，sstream 的函数返回的是 char * 类型的字符串，而 stratream 的函数返回的是 std::string 类型的字符串。
- strstream 已被C++标准宣称为 deprecated。

第14章　流及类库

◆14.7 字符串流

- sstream 包括：
  - 输入流类 istringstream，只能从 std::string 中读取数据
  - 输出流类 ostringstream，只能写数据到 std::string 中
  - 输入输出流类 stringstream（将istringstream和ostringstream合到一起）
  - 输入输出流类 stringbuf
- strstream 包括 < deprecated >：
  - 输入流类 istrstream，只能从char *缓冲区中读取数据
  - 输出流类 ostrstream，只能写数据到char *中
  - 输入输出流类 strstream（将istrstream和ostrstream合到一起）
  - 输入输出流类 strstreambuf

第14章　流及类库

◆14.7 字符串流

```
#include <iostream>
#include <sstream>
#include <string.h>
using namespace std;
int main(int argc, char *argv[ ])
{
    stringstream ss;
    string str = "abc";
    ss << str << 12345 << "  123";    //ss:  abc12345  123
    string s;  int n;
    ss >> s >> n;     //s = abc12345,  n = 123
    cout << ss.str() << ": " << s << "," << n;   //abc12345, 123
    return 0;
}
```

第14章　流及类库

◆14.7 字符串流

```
#include <iostream>
#include <sstream>
#include <string.h>
using namespace std;

template <typename T>    //T是简单类型 (int、float、double、char * 等)
char *print(T e, char *s)   //将e转换为字符串并保存到缓冲区s中
{
    stringstream ss;
    ss << e;
    string str = ss.str();
    strcpy(s, str.c_str());
    return s;
}
```

如果T是复杂类型（如自定义的类）怎么办?

第14章　流及类库

◆14.7 字符串流

如果T是复杂类型, 怎么办?

重载 operator<<()

```
#include <iostream>
#include <sstream>
using namespace std;

stringstream &operator<<(stringstream &ss, const int a[])
{  //int 数组以 0 结尾
   for(int k = 0; a[k] != 0; k++) ss << a[k] << " ";
   return ss;
}

int main() {
   stringstream  ss;
   int a[] = { 5, 4, 12, 34, 0 };
   ss << a;
   cout << ss.str();
}
```

第14章　流及类库

◆14.7 字符串流

如果不使用字符串流，则只能使用 typeid 运算符
(typeid()返回const type_info &)

```
template <typename T>    //T是简单类型 (int、float、double、char * 等)
char *print(T e, char *s)   //将e转换为字符串并保存到缓冲区s中
{
    if(typeid(T)==typeid(int)) sprintf(s,"%d", e);
    if(typeid(T)==typeid(float)) sprintf(s,"%f", e);
    if(typeid(T)==typeid(char *)) sprintf(s,"%s", e);
    if(typeid(T)==typeid(char)) sprintf(s,"%c", e);
    ... ...
    return s;
}
```