面向对象程序设计(OOP)

C++(第10章：C++流和I/O系统)

福州大学·软件学院·软件工程系
王灿辉(wangcanhui@fzu.edu.cn)

第10章：C++流和I/O系统

概述

- C中的I/O函数的数据类型是由用户负责检查的，而不是由编译器检查的，所以很容易引入错误。例如：

```
void func(int i,float f)
   {printf("%d %d\n",i,f)}
```

编译器检测不出错误！

第10章：C++流和I/O系统

概述：C++ I/O流的优点

- 它是类型安全的
- 利用运算符重载，用户自定义的类型也可以同内部类型一样进行输入/输出。
- 流的书写格式简单、清晰可以增进程序的可读性。
- C++也可以使用C的I/O函数，但最好用流来进行输入/输出。

第10章：C++流和I/O系统

概述：运算符<<和>>的优先级

```
#include <iostream>
using namespace std;
void main() {
   int i=016,j=03;          //八进制数
   cout<<i+j<<endl;         //输出17
   cout<<(i&j)<<endl;       //输出2
   //(i&j)的括号不能少，否则编译报错！
   cout<<j<<','<<j++<<endl; //输出4,3
}
```

第10章：C++流和I/O系统

用getline函数读含空白的串

```
#include <iostream>
#include<conio.h> //getch();
using namespace std;
void main() {
  char str[80];
  cout << "Enter your string: ";
  cin.getline(str, 79);
  cout << str << '\n';
  cout<<"按任意键继续...";getch();
}
```

第10章：C++流和I/O系统

概述：I/O流和缓冲区

- C++中把数据之间的传输操作称为流。

注意：输入和输出是相对于正在编写的程序而言的。

第10章：C++流和I/O系统

I/O操作

- 在进行I/O操作时，首先是打开操作，使流和文件发生联系，建立联系后的文件才允许数据流入或流出，输入或输出结束后，使用关闭操作使文件与流断开联系。
- 文件的打开和关闭是通过使用fstream类的成员函数open和close来实现的。

第10章：C++流和I/O系统

预定义的C++流

- C++包含了几个预定义流：cin(标准输入流)、cout(标准输出流)、cerr(链接到标准输出的非缓冲流)、clog(链接到标准输出的缓冲流)。cerr和clog一般用于输出程序调试或错误信息。
- C++还打开标准流的宽(16位)字符版本：wcin、wcout、wcerr、wclog。

第10章：C++流和I/O系统

运算符<<和>>的重载

- 只能用友员函数或普通函数(不能用成员函数)重载抽取运算符>>和插入运算符<<，重载函数格式为：

```
istream &operator>>(istream &is,T &obj)
  {/*…*/ return is;}
ostream &operator<<(ostream &out,const T &o)
  {/*…*/ return out;}
```

第10章：C++流和I/O系统

实例：重载抽取运算符>>

```
istream &operator>>(istream &is,Date &d) {
   char d_str[80];
   is >> d_str; //读入日期串
   d=d.stringToDate(d_str);//类型转换
   return is;
}
```

第10章：C++流和I/O系统

实例：重载插入运算符<<

```
ostream &operator<<(ostream &out,const Date &d)
{
    out << "{" << d.year << "."
        << d.month << "."
        << d.day << "}";
    return out;
}
```

第10章：C++流和I/O系统

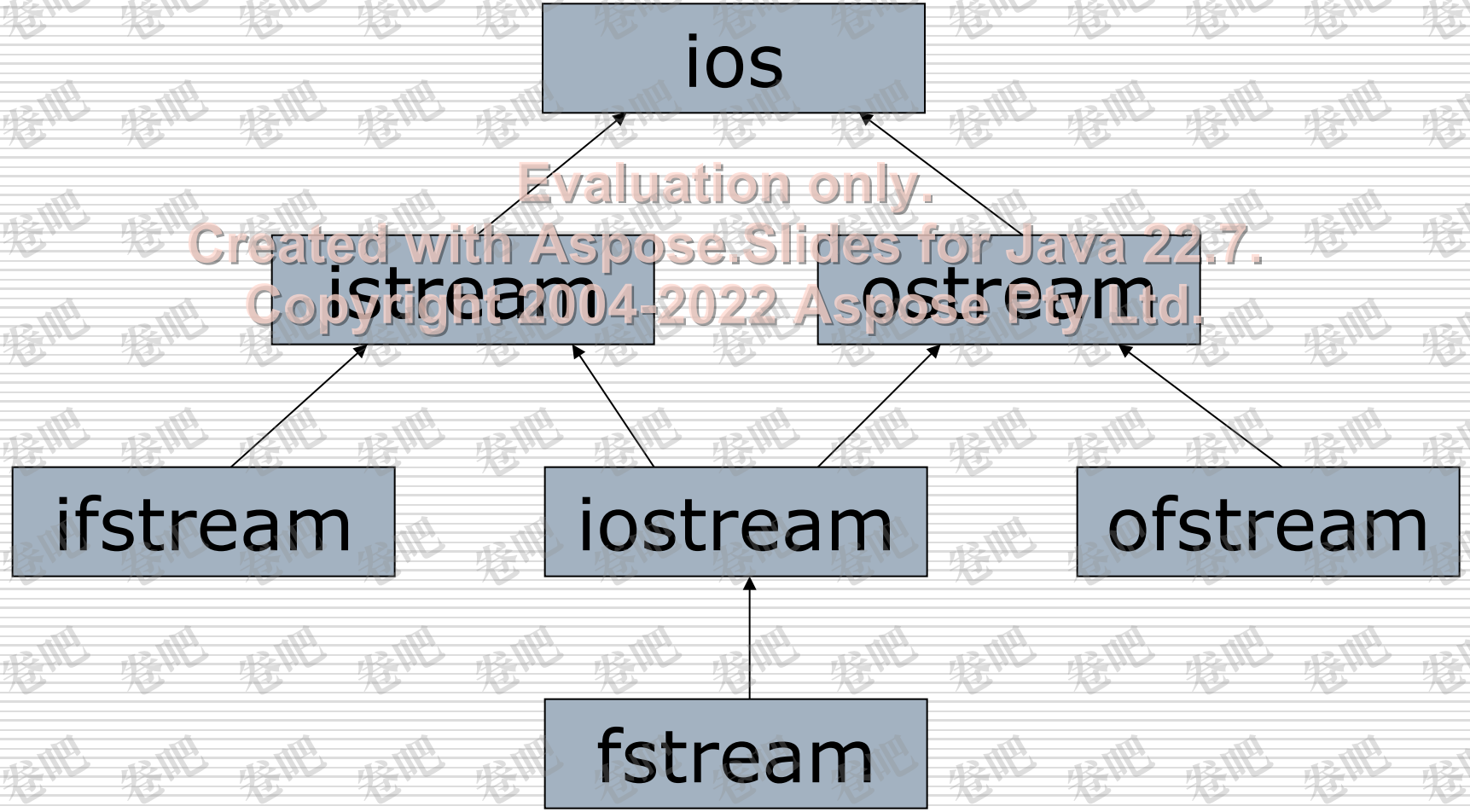
I/O流类模板：基于字符的类名称

- C++ I/O系统基于字符的类名称是：streambuf、ios、istream、ostream、iostream、fstream、ifstream、ofstream。
- 实际上这些类名称是进行了如下的定义(Iosfwd)：

```
typedef basic_ios<char,char_traits<char>> ios;
//……
typedef basic_ios<w_char_t,char_traits<w_char_t>> wios;
//……
```

第10章：C++流和I/O系统

C++ I/O流类层次图(略)



第10章：C++流和I/O系统

格式化I/O

- 可以使用两种方法来控制数据的(输入和输出)格式，第一种方法使用ios类(模板)的成员函数[cout.width(10);]，第二种方法使用称为操控符的特殊函数类型[cout<<setw(10)<<123.456;]。
- 宽度设置<输出项长度，设置无效！

第10章：C++流和I/O系统

格式化I/O：使用ios类的成员函数

- ios类(模板)的格式控制函数(Ios.h)：

```
inline long flags(long _l);// //设置标志
inline long setf(long _f,long _m);//设置标志位
inline long setf(long _l); //设置标志位
inline long unsetf(long _l); //清除标志
inline int width() const;//获得当设置宽度
inline int width(int _i); //设置字段宽度
inline char fill(char _c);//设置填充字符
inline int precision(int _i);//浮点数精度
//……,fill等函数返回原值,可据此恢复原设置!
```

第10章：C++流和I/O系统

格式化I/O：使用ios类的成员函数

- ios类(模板)的格式状态(Ios.h)：

```
enum {
    skipws=0x0001,
    left=0x0002,right=0x0004,internal=0x0008,
    dec=0x0010,oct=0x0020,hex=0x0040,
    showbase=0x0080,showpoint=0x0100,
    uppercase=0x0200,showpos=0x0400,
    scientific=0x0800,fixed=0x1000,
    unitbuf=0x2000,    //……
};
```

第10章：C++流和I/O系统

格式化I/O：使用ios类的成员函数

```
#include <iostream>
using namespace std;
void main(){
   cout<<"字段宽度["<<cout.width()<<"]\n";
   cout<<"填充字符["<<cout.fill()<<"]\n";
   cout<<"浮点数精度["<<cout.precision()<<"]\n";
   cout<<"标志(flags)["<<cout.flags()<<"]\n";
   cout<<"I/O状态["<<cout.rdstate()<<"]\n";
}//输出：[0][ ][6][513(=0x201)][0]
```

第10章：C++流和I/O系统

格式化I/O：使用ios类的成员函数

```
#include <iostream>
using namespace std;
void main() {
  //设置showpos和scientific标志
  cout.setf(ios::showpos);/*等价于:
   cout.flags(cout.flags()|ios::showpos)*/
  cout.setf(ios::scientific);
  cout << 123 << " " << 123.23 << "\n";
  //输出：+123  +1.232300e+002
}
```

第10章：C++流和I/O系统

格式化I/O：使用ios类的成员函数

```
#include <iostream>
using namespace std;
void main(){//默认右对齐setf(ios::right)
  cout.precision(2); //浮点数精度，一直起作用
  cout.fill('#'); //用#填充,一直起作用
  cout.width(10); //字段宽度为10,仅对下一项起作用
  cout << 123 << " ";
  cout.width(10);
  cout << 123.23 << "\n";
}//输出：########123 ##1.2e+002
```

第10章：C++流和I/O系统

用width限制输入字符数

```
#include <iostream>
using namespace std;
void main() {//假定输入1234567890
  char buf[4],str[80];
  cin.width(4);      //最多读入3个字符
  cin>>buf;
  cout<<buf<<endl; //输出123
  cin>>str;
  cout<<str<<endl; //输出4567890
}
```

第10章：C++流和I/O系统

格式化I/O：使用ios类的成员函数

- 可以使用flags()和unsetf()来设置、清除格式标志。可以恢复默认格式。
- 可以用dec, oct, hex按不同进制输出整数值，用showbase显示基数等。
- 可以用right, left, internal调整字段的输出对齐方式。

第10章：C++流和I/O系统

格式化I/O:使用操控符

- 使用操控符必须包含头文件：<Iomanip>

  setiosflags(long _l):设置字段标志

  resetiosflags(long _l):清除字段标志

  setfill(int _m):设置填充字符

  setprecision(int _p):设置浮点数精度

  setbase(int _b):设置基数

  setw(int _w):设置字段宽度,仅对下一项起作用

第10章：C++流和I/O系统

格式化I/O:使用操控符

```
#include <iostream>
#include <iomanip>
using namespace std;
void main() {
  cout<<setprecision(7)<<1234.5678<<endl;
  //输出：1234.568
  cout<<setw(20)<<setfill('@')<<"Hello World!"<<endl;
  //输出：@@@@@@@@Hello World!
}
```

第10章：C++流和I/O系统

格式化I/O:使用操控符

```
#include <iostream>
#include <iomanip>
using namespace std;
void main() {
  cout<<setiosflags(ios::showpos)<<
        setiosflags(ios::scientific)<<
        123<<" "<<123.23<<endl;
  //输出：+123 +1.232300e+002
}
```

第10章：C++流和I/O系统

格式化I/O:使用操控符

```
#include <iostream>
#include <iomanip>
using namespace std;
void main() {
   bool t=true,f=false;
   cout<<t<<’ ’<<f<<endl; //输出： 1 0
   cout<<setiosflags(ios::boolalpha)<<
     t<<’ ’<<f<<endl;//输出：true false
   cout<<resetiosflags(ios::boolalpha)<<
     t<<’ ’<<f<<endl; //输出：1 0
}
```

第10章：C++流和I/O系统

格式化I/O:使用操控符

- I/O操控符参见(Ios)：
  boolalpha(用符号形式输出真和假)、internal、right、left、dec、oct、hex、fixed、scientific、showbase、showpoint(输出尾部的0)、showpos、skipws(跳过空白)、unitbuf(每次输出之后刷新)、uppercase、endl(输出换行并刷新)、ends(‘\0’)、flush(刷新)

第10章：C++流和I/O系统

格式化I/O:使用操控符

- 可以创建自己的操控符函数，例如：

  创建一个无参的输出操控符

  创建一个无参的输入操控符

- 创建有参的操控符函数函数参见其他书籍，如：(美)Bjarne Stroustrup著，裘宗燕 译，C++程序设计语言，北京：机械工业出版社，2002年7月。

第10章：C++流和I/O系统

文件I/O

- 文件是存储在磁盘、磁带等外部存储设备上的数据的集合，每一个文件都必须有一个唯一的文件名称。在使用文件前必须首先打开文件，使用完毕后必须关闭文件。对文件的操作是由文件流类来完成的。

第10章：C++流和I/O系统

文件I/O

- 对文件的操作过程可按以下四步进行：即定义文件流类的对象、打开文件、对文件进行读写操作、关闭文件。
- 可以在调用构造函数时同时打开文件，当文件流对象离开它的作用域时会自动关闭文件(但最好显式关闭文件)。
- 所以最简单方式是：构造、读写两步。

第10章：C++流和I/O系统

文件I/O：定义文件流对象

- 在C++中，打开文件的方式是将文件链接到流(输入ifstream、输出ofstream、输入/输出fstream)。例如：

```
ifstream in;    //输入
ofstream out;   //输出
fstream  both; //输入/输出
```

第10章：C++流和I/O系统

文件I/O：打开文件

- 定义了文件流对象后，就可以利用其成员函数open()打开文件。该成员函数的格式为：

```
void open(const unsigned char
                *filename,int mode);
```

其中：filename为文件名，mode为文件打开方式。

第10章：C++流和I/O系统

文件I/O：文件打开方式

- enum open_mode {in=0x01,out=0x02,
  ate=0x04,app=0x08,trunc=0x10,
  nocreate=0x20,noreplace=0x40,
  binary=0x80};//ate:at end
- 如果未指明以二进制方式打开文件，则默认以文本方式打开。
- 对ifstream流打开方式默认为ios::in，对ofstream流默认为ios::out|ios::trunc，fstream流默认为ios::in|ios::out(可读写)

第10章：C++流和I/O系统

文件I/O：文件打开方式

- 文件打开方式可以用“|”组合起来，例：
  ios::in|ios::out：以读写方式打开
  ios::in|ios::binary：以二进制读方式打开文件
  ios::truc|ios::binary：以二进制写方式打开文件，若文件存在则清除文件内容，若文件不存在则创建新文件。

第10章：C++流和I/O系统

文件I/O：文件属性

- 成员函数open()在有些版本的C++中支持第3个参数：文件属性，取值为：

  0一般文件

  1只读文件

  2隐藏文件

  3系统文件　//VC++编译器不支持！

第10章：C++流和I/O系统

文件I/O：文件关闭

- 文件读写结束后应及时调用成员函数close()关闭文件。该函数没有参数也没有返回值。
- 在当文件流对象离开它的作用域时会自动关闭文件(但最好显式关闭文件)。

第10章：C++流和I/O系统

文件I/O：文件的读写操作

- 文件读：用运算符>>，使用文件流类的get(读1个字符),getline(读取多个字符),read(读数据块)等成员函数。
- 写文件：用运算符<<，使用文件流类的put(写1个字符),write(写数据块)等成员函数。

第10章：C++流和I/O系统

文件I/O：写文本文件

```
#include <iostream>
#include <fstream>
using namespace std;
void main() {
  ofstream out("test.txt");//写打开，文本文件
  if(!out)//等价于：if(!out.is_open())
    {cout<<"Cannot open file.\n";return;}
  out << 10 << " " << 123.23 << "\n";
  out << "This is a short text file.\n";
  out.close();
}//用运算符<<写入文件
```

第10章：C++流和I/O系统

文件I/O：写二进制文件

```
#include <iostream>
#include <fstream>
using namespace std;
void main() {
  ofstream out;
  out.open("test.txt",ios::out|ios::binary);
  //使用open()成员函数，写打开，二进制文件
  if(!out) {cout<<"Cannot open file.\n";return;}
  out << 10 << " " << 123.23 << "\n";
  out << "This is a short text file.\n";
  out.close();
}//用运算符<<写入文件
```

第10章：C++流和I/O系统

文件I/O：读文本(二进制)文件

- 可以用>>运算符从一个文件读入数据

```
//……
ifstream in("test.txt");
if(!in) {//……}
in >> i;  in >> f;  in >> ch;
in >> str;//遇到空白或回车结束！
//……
```

第10章：C++流和I/O系统

读/写二进制(文本)文件

- 文件操作成员函数：get(char &ch); put(char ch); getline(char *pch,int count,char delim='\n');read(char *pch, int count);write(const char *pch,int count);gcount()//计算已读的字符数
- 使用get()函数读取文件数据
- 使用put()函数写（二进制）文件
- 用read()和write()读写二进制数据块

第10章：C++流和I/O系统

其他函数

- get有多个重载版本，get和getline的主要区别在于是否读入换行符。
- eof()判文件结束，flush()刷新流，peek()获得下一字符，但不从流中删除；unget()退回最后读入的字符；putpack(char c)将刚从流读取的字符返回流中；ignore(int)跳过n个字符;ostream *tie(ostream *s);把输入流链接到一个输出流上;等。

第10章：C++流和I/O系统

文件I/O：文件的随机访问

- 随机文件提供了在文件中来回移动文件指针和非顺序读写文件的能力。

  seekg(long pos);seekg(long off,seekdir); tellg(); seekp(long pos); seekp(long off,seekdir);tellp();

- 其中：pos为绝对位置，off为偏移，dir为：ios::cur、ios::beg、ios::end三者之一。

第10章：C++流和I/O系统

文件I/O：文件的随机访问

- 文件的随机写入

```
out.seekp(atol(argv[2]),ios::beg);
out.put('X');
```

- 从文件的给定位置开始显示

```
in.seekg(atol(argv[2]));
in.seekg(-atol(argv[2])+1,ios::cur);
while(in.get(ch)) cout << ch;
```

- 综合实例：文件的比较

第10章：C++流和I/O系统

检查I/O状态

- enum io_state{goodbit=0x00,eofbit=0x01,failbit=0x02,badbit=0x04}; //Ios.h
- 相关函数:rdsate()返回I/O状态;bad();eof();fail();good();四个检查I/O状态的函数，返回类型均为bool。clear(iostate flags=ios::goodbit);设置/清除I/O状态。

第10章：C++流和I/O系统

其他I/O流：略

- 字符串流类:stringstream，istringstream，ostringstream。对应于C的sprintf()和scanf()等函数。
- 流缓冲区类:streambuf，filebuf，stringbuf。

本章内容讲授到此结束！

福州大学·软件学院·软件工程系
王灿辉(wangcanhui@fzu.edu.cn)