第 5 章
类和对象应用

5.1 基于项目的多文件管理

基于项目的多文件管理基于如下的规则：

(1) 将类的设计与类的使用分离，即类定义与 main 函数不在一个文件中。

(2) 将类的声明和类的成员函数实现分离，即类定义与成员函数定义不在一个文件中。

这么做的好处是：便于分工合作；便于软件的维护。

在 Dev-Cpp 中选择“文件”→“新建项目”菜单命令，将弹出如图 5.1 所示的对话框。

图 5.1 “新项目”对话框

图 5.1 中项目文件的扩展名是 dev，项目类型是控制台应用程序，项目语言是 C++ 项目，项目名称是 L5.1。注意将新建的项目文件保存到一个新建的子目录。

[**例 5.1**] 设计一个圆类，并计算圆的面积。要求基于项目的多文件管理方式。

```
//类的定义(Circle.h 文件)
class Circle
{
public:
    Circle(double a=0);
    double Area();
protected:
    double r;
};
```

```
//类的成员函数实现(Circle.cpp 文件)
#include "Circle.h"
Circle::Circle(double a)
{  r=a;  }
double Circle::Area()
{  return 3.14 * r * r;  }
//主函数(main.cpp 文件)
#include <iostream>
#include "Circle.h"
using namespace std;
int main()
{   Circle c(5);
    cout<<c.Area()<<endl;
    return 0;
}
```

输出

```
78.5
```

基于项目的多文件管理步骤(以例 5.1 的圆类定义和对象使用为例)如下：

(1) 创建一个控制台类型的项目 L5.1,带一个 main.cpp 文件,其内容是 main 函数。

(2) 在 L5.1 项目中为 Circle 类创建两个文件(circle.h 和 circle.cpp)。

① 手动添加代码：在 Dev-Cpp 左侧“项目管理”面板的项目名 L5.1 上右击,在快捷菜单中选择 New File(新建文件)命令,添加两个新文件,如图 5.2 所示。

将新建的两个文件保存到与项目文件同一个子目录下,取名 Circle.h 和 Circle.cpp,最后将例 5.1 有关代码复制到对应文件中即可。

② 系统自动加了框架代码。在 Dev-Cpp 左侧“查看类”面板中右击在快捷菜单中选择“新建类”命令,如图 5.3 所示。或者在“文件”菜单选择“新建”→“新建类”命令。

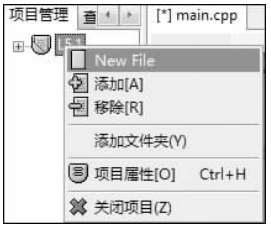

图 5.2　新建两个文件

图 5.3　新建一个类

系统将弹出如图 5.4 所示的对话框。

在图 5.4 中输入类名 Circle,建议勾选“创建构造函数”复选框。可以选择性地输入构造函数参数 double a,当然不输入也可以。单击右侧的更改按钮“…”可以更改文件名,如果不改,则使用默认文件名 Circle.cpp 及 Circle.h。单击“创建”按钮之后,可以在 Dev-Cpp 左侧的“项目管理”面板看到系统自动创建了两个文件——Circle.cpp 及 Circle.h,如图 5.5 所示。最后将例 5.1 有关代码复制到对应文件中即可。注意这两个文件要与项目文件保存在

同一个子目录下。

图 5.4 “新建类”对话框

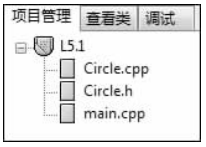

图 5.5 项目 L5.1 的文件

对于大型复杂程序一般采用基于项目的多文件管理,也就是说一个项目由多个文件构成。

[**例 5.2**] 将第 4 章课堂练习的完整代码以基于项目的多文件管理形式组织。

```
//L5_2.cpp
#include <iostream>
#include <string>
using namespace std;
class student          //以下注释中的(1)~(8)对应第 4 章课堂练习的问题(1)~(8)的相关代码
{
public:
    void output();
    void set(const string &p,int i,double x,double y) //(4)
    {  name=p; num=i; m_c=x; m_math=y;  }
    double jstotal()                                   //(5)
    {  return m_c+m_math;  }
    student(const string &p="wang",int i=1,double x=0,double y=0)  //(7)
    {  name=p; num=i; m_c=x; m_math=y;  }
private:
    string name;                                       //姓名
    int num ;                                          //学号
    double m_c;                                        //语文成绩
    double m_math;                                     //数学成绩
};
void student::output()
{
    cout<<"姓名:"<<name<<","<<"学号:"<<num<<","
    <<"语文:"<<m_c<<","<<"数学:"<<m_math <<endl;   //(4)
}
```

```
int main()
{
   //string x("wang");                                    //等价于 string x="wang";
   //student s;                                           //(1)定义对象
   //s.set(x,1,85.5);                                     //(2)设置学生数据
   //s.output();                                          //(3)输出学生数据
   //s.set(x,1,92.1);                                     //(4)修改学生成绩数据
   //cout<<"总成绩:"<<s.jstotal()<<endl;                  //(6)
   student t;                                             //(8)定义对象 t
   t.output();                                            //(8)输出学生数据
   cout<<"总成绩:"<<t.jstotal()<<endl;                    //(8)
   student z("liuxiang",2,100,95);                        //定义对象 z
   z.output();
   cout<<"总成绩:"<<z.jstotal()<<endl;
   return 0;
}
```

与例 5.1 一样，新建一个 L5.2 项目，该项目有 3 个文件，具体作用和内容如下所示：

```
//student.h: 类定义
#include <string>
using namespace std;
class student
{
public:
    void output();
    void set(const string &p,int i,double x,double y);
    double jstotal();
    student(const string &p="wang",int i=1,double x=0,double y=0);
protected:
    string name;                                          //姓名
    int num ;                                             //学号
    double m_c;                                           //语文成绩
    double m_math;                                        //数学成绩
};

//student.cpp: 成员函数定义
#include "student.h"
#include <iostream>
#include <string>
using namespace std;
void student::output()
{   cout<<"姓名:"<<name<<","<<"学号:"<<num<<","
    <<"语文:"<<m_c<<","<<"数学:"<<m_math <<endl;          //(4)
}
void student::set(const string &p,int i,double x,double y)  //(4)
```

```
{  name=p; num=i; m_c=x; m_math=y;  }
double student::jstotal()                                    //(5)
{  return m_c+m_math;  }
student::student(const string &p,int i,double x,double y) //(7)
{  name=p; num=i; m_c=x; m_math=y;  }

//main.cpp: 类测试
#include <iostream>
#include <string>
#include "student.h"
using namespace std;
int main() {
    //string x("wang");                                      //等价于 string x="wang";
    //student s;                                             //(1)定义对象
    //s.set(x,1,85.5);                                       //(2)设置学生数据
    //s.output();                                            //(3)输出学生数据
    //s.set(x,1,92.1);                                       //(4)修改学生成绩数据
    //cout<<"总成绩:"<<s.jstotal()<<endl;                    //(6)
    student t;                                               //(8)定义对象 t
    t.output();                                              //(8)输出学生数据
    cout<<"总成绩:"<<t.jstotal()<<endl;                      //(8)
    student z("liuxiang",2,100,95);                          //定义对象 z
    z.output();
    cout<<"总成绩:"<<z.jstotal()<<endl;
    return 0;
}
```

例 5.2 按项目的方式管理多个文件时，出现 iostream、string 等头文件被重复包含多次，但是没有出现什么重定义错误，这是因为系统这些头文件中的内容包含 ifndef 或者 #pragma once，这两个宏就是为了避免同一个头文件被包含多次时，这样就不会出现重定义的错误。

#ifndef 方式示例：

```
#ifndef __AFILE_H__
#define __AFILE_H__
…                                                            //一些声明语句
#endif
```

#ifndef 的方式依赖于宏名字不能冲突，这不仅可以保证同一个文件不会被包含多次，也能保证内容完全相同的两个文件不会被同时包含。当然，这种方式的缺点就是如果不同头文件的宏名不小心“撞车”，可能就会导致头文件明明存在，编译器却硬说找不到声明的状况。

#pragma once 方式示例：

```
#pragma once
…                                                            // 一些声明语句
```

#pragma once 则由编译器保证同一个文件不会被包含多次。注意这里所说的“同一个文件”是指物理上的一个文件，而不是指内容相同的两个文件。它带来的好处是，你不必再费劲想个宏名了，当然也就不会出现宏名冲突引发的奇怪问题。这种方式的缺点就是如果某个头文件有多份副本，就不能保证它们不被重复包含。当然，相比宏名冲突引发的“找不到声明”的问题，重复包含更容易被发现并修正。

可见，对于#ifndef 方式和#pragma once 方式都能够支持的编译器，两者并没有太大的区别，但是两者仍然有一些细微的区别。#ifndef 方式的移植性好，#pragma once 方式可以避免名字冲突。

5.2　文件与流操作

文件的处理由 3 个步骤组成：打开文件、读写文件、关闭文件。

C++ 把每一个文件都看成一个有序的字节流，对文件的操作可采用与输入输出流相关的方法，要加对应的文件包含命令：#include <fstream>。

(1) 打开读文件有两种方法：

① 先建立文件流对象，再调用成员函数 open 将它与某一个文件关联。例如：

```
ifstream infile;                                    //输入文件流对象
infile.open("a.txt");                               //可省略打开模式,默认是读文本文件
infile.open("a.dat",ios::binary);                   //打开读二进制文件
```

② 在建立文件流对象的同时通过构造函数打开文件。例如：

```
ifstream infile ("a.txt");                          //打开读文本文件
ifstream outfile("a.dat",ios::binary);              //打开读二进制文件
```

测试文件是否被正确打开的方法如下：

```
if ( ! infile)                  //等价于 if(!infile.is_open())或 if(!infile.fail())
{   …                           //处理文件打开失败情况的代码
}
```

(2) 关闭读文件，涉及成员函数 close。例如：

```
infile.close( );
```

(3) 打开写文件有两种方法：

① 先建立文件流对象，再调用成员函数 open 将它与某一个文件关联。格式如下：

```
fstream iofile;                                     //输入输出文件流对象
iofile.open(文件名,文件打开方式);                    //不能省略
```

② 在建立文件流对象的同时通过构造函数来打开文件。文件打开方式不能省略。格式如下：

```
fstream iofile(文件名, ios::in);                    //读文本文件
fstream iofile(文件名, ios::out);                   //写文本文件
```

```
fstream iofile(文件名, ios::in|ios::binary);   //读二进制文件
fstream iofile(文件名, ios::out|ios::binary);  //写二进制文件
```

测试文件是否被正确打开的方法如下：

```
if ( ! iofile)                  //等价于 if(!iofile.is_open())或 if(!iofile.fail())

{   …                           //处理文件打开失败情况的代码
}
```

(4) 关闭写文件，涉及成员函数 close。格式如下：

```
iofile.close( );
```

(5) 文本文件的读写。

① 使用插入与提取运算符对文本文件进行读写：>>为读文件，<<为写文件。

② 使用类成员函数对文件流进行字符操作。

get 为读，put 为写：一次读写一个字节，函数原型如下：

```
istream& get(char& rch);
ostream& put(char ch);
```

getline：一次读一行，函数原型如下：

```
istream& getline(char * pch, int nCount, char delim = '\n');
```

文件打开方式如表 5.1 所示：

表 5.1 文件打开方式

打开方式	说明
ios::app	将所有输出写入文件末尾
ios::ate	打开文件以便输出，并移到文件末尾(通常用于添加数据) 数据可以写入文件中的任何地方
ios::in	打开文件以便输入
ios::out	打开文件以便输出
ios::trunc	删除文件现有内容(是 ios::out 的默认操作)
ios::binary	用二进制而不是文本模式打开文件
ios::nocreate	如果文件不存在，则文件打开失败
ios::noreplace	如果文件存在，则文件打开失败

[**例 5.3**] 写文本文件示例。

```
//L5_3.cpp
#include <iostream>
#include <fstream>
using namespace std;
int main()
```

```
{
    ofstream outfile("d:\\grade.txt");
    //写文本文件,等价于 fstream outfile("d:\\grade.txt",ios::out);
    if(!outfile)
    {
        cout <<"文件打开失败!"<<endl;
        return 1;
    }
    outfile <<"程序设计" <<"  " <<95 <<endl;
    outfile <<"大学英语" <<"  " <<90.5 <<endl;
    outfile <<"高等数学" <<"  " <<93 <<endl;
    outfile <<"普通物理" <<"  " <<87.5 <<endl;
    outfile.close();
    return 0;
}
```

输出如图 5.6 所示。

图 5.6　grade.txt 文件内容

［**例 5.4**］　读文本文件示例。

```
//L5_4.cpp
#include <iostream>
#include <fstream>
using namespace std;
int main()
{   ifstream infile("d:\\grade.txt");
    //读文本文件,等价于 fstream infile("d:\\grade.txt",ios::in);
    if(!infile)
    {   cout <<"文件打开失败!"<<endl;
        return 1;
    }
    char course[20];
    float score;
    infile >>course >>score;
    cout <<course <<"  " <<score <<endl;
    infile >>course >>score;
    cout <<course <<"  " <<score <<endl;
    infile >>course >>score;
    cout <<course <<"  " <<score <<endl;
    infile >>course >>score;
    cout <<course <<"  " <<score <<endl;
    infile.close();
    return 0;
}
```

输出

```
程序设计语言    95
```

```
大学英语        90.5
高等数学        93
普通物理        87.5
```

［**例 5.5**］ 使用成员函数 get 完成读文本文件示例。

```
//L5_5.cpp
#include <iostream>
#include <fstream>
using namespace std;
int main()
{   char ch;
    int count=0;                                    // 计数器
    ifstream infile("d:\\grade.txt");
    //读文本文件,等价于 fstream infile("d:\\grade.txt",ios::in);
    if(!infile)
    {
        cout <<"文件打开失败"<<endl;
        return 1;
    }
    while(!infile.eof( ))
    {   infile.get(ch);                             // 从文件流中读入下一个字符
        cout<<ch;                                   // 屏幕输出从文件中读入的字符
        if(ch>='0' && ch<='9') count++;             // 若是数字字符,计数器加 1
    }
    cout<<"文件中共有数字字符:"<<count<<"个。"<<endl;
    infile.close();
    return 0;
}
```

输出

```
程序设计语言          95
大学英语              90.5
高等数学              93
普通物理              87.5
文件中共有数字字符:10 个
```

［**例 5.6**］ 打开一个由若干个整数组成的文本文件 number. txt,找出其中所有的素数并存入另一个文本文件 prime. txt 中。

```
//L5_6.cpp
#include <iostream>
#include <fstream>
using namespace std;
int isprime(int a)                                  // 素数判断函数
{   for(int i=2; i<=a/2; i++)
        if(a%i ==0) return 0;
```

```
    return 1;
}
int main()
{   ifstream infile("number.txt");
    ofstream outfile("prime.txt");
    if(!infile || !outfile)
    {   cout <<"文件打开失败!"<<endl;
        return 1;
    }
    int num;
    while(!infile.eof() )
    {   infile>>num;
        if(isprime(num)) outfile<<num<<"  ";
    }
    infile.close();
    outfile.close();
    return 0;
}
```

文件 number.txt 的内容：23 2 7 4。文件 prime.txt 的内容：23 2 7。

(6) 二进制文件的读写。

二进制文件以位(b)为单位，整个文件是由 0 和 1 组成的无格式的原始数据序列。二进制方式下的输入输出过程中，系统不对数据进行任何转换。文本文件以字节(B)为单位，整个文件实际保存的是一串 ASCII 字符。可用文字处理器进行编辑。在文本方式下的输入输出过程中，系统进行字符转换。

二进制文件的读写比文本文件复杂，不能用插入或提取符。

① put 函数向流写一个字符，其原型是 ofstream &put(char ch)，使用也比较简单，例如 file1.put('c')；就是向流写一个字符'c'。

② get 函数从流读字符或者字符串，比较灵活，有 3 种常用的重载形式。

③ 读写数据块。

要读写二进制数据块，使用成员函数 read 和 write，它们的原型如下：

```
read(unsigned char * buf,int num);          //从文件中读取 num 个字符到 buf 指向的缓存中
write(const unsigned char * buf,int num); //将 buf 指向的缓存中的 num 个字符写到文件中
```

[**例 5.7**]　二进制文件复制示例。

```
//L5_7cpp
#include <iostream>
#include <fstream>
using namespace std;
int main()
{   char s[50], d[50];
    cout<<"请输入准备复制的文件名(含后缀名):";
    cin>>s;
```