由一个简单的程序谈起－－之五（精华）

江苏 无锡 缪小东

本篇和大家讨论该“学生管理系统”的核心部分－－一个特殊的数据结构。该数据结构的名称为DLinkedStuCollection 即一个双向的链表，可以完成向前向后的遍列，同时还能在内存中没有数据时自动从文件系统加载记录，还可以在内存中的数据到达某个极限的时候将某些记录保存到文件了！

```
import java.io.Serializable;
import java.util.*;
import java.io.*;

public class DLinkedStuCollection implements DOA{
    DOAFileUtil util = new DOAFileUtil();        //一个比较重要的内部类，主要完成和文件系统的交互
                                                 //其实完全可以放在本类中，主要为了隐藏和文件交互的细节
    public static final int    DEFAULTSIZE = 400 ;        //内存中最多保存记录的数目，可以自行更改
    public static final int STU_NUM = 50;                 //每个文件中最多保存记录的数目
                                                          //这两个参数可以设计成用户可以设置的，省略了！
    private int capacity    ;                             //内存可以包含记录的容量
    private int size = 0 ;                                //内存中已经有的记录的数目
    int cursor = 0 ;                                      //当前指针的位置，用于记录游标的位置
    StuNode curr = null ;                                 //当前的节点，和游标的位置对应

    StuNode header = new StuNode(null,null,null);         //一个双向链表的标准的头

    private static class StuNode /**implements Serializable*/ {   //学生的节点，保存学生记录和前后指针
        StuNode prev ;                                    //前一个节点的指针
        Student elem ;                                    //本节点的实际内容
        StuNode next ;                                    //下一个节点的指针
        StuNode(StuNode prev,Student elem,StuNode next ){
            this.prev=prev ;
            this.elem=elem;
            this.next=next ;
        }
    }               //这是一个双向链表中典型的节点的构造方法

    public DLinkedStuCollection(int initialCapacity){          //构造器，控制链表的最大容量
        header.prev=header.next=header;                        //默认时头连接它本身！为什么？仔细研究吧！
        capacity = Math.max(initialCapacity,DEFAULTSIZE);      //确认最小容量为 400
        size = 0 ;                                             //刚开始肯定没有元素
        cursor = 0 ;                                           //刚开始游标的位置肯定为 0
        curr = header;                                         //刚开始当前节点肯定为头了
    }

    public DLinkedStuCollection(){                             //默认构造器，使用默认的参数呗
```

```
        this(DEFAULTSIZE );
}

public void add(Student stu)throws Exception{                    //在当前位置的后面添加一个记录
        if(size>=capacity){                                      //增加的时候，超过最大容量时会保存部分元素
                if(cursor < (capacity/2)){                       //游标在链表的前端，保存后端 50 个元素
                        util.saveBack();                         //保存后端，是在内部类中完成的
                }else{
                        util.saveHeader();                       //保存前端，是在内部类中完成的
                }
        }
        StuNode added = new StuNode(curr,stu,curr.next);         //创建一个新的节点
        curr.next.prev=  added;                                  //注意节点指针的变化哦！顺序绝对不要乱！
        curr.next=added;
        curr=curr.next;                                          //以上是在当前节点的后面添加一个节点
        cursor++;
        size++;                                  //这个过程游标、元素的个数以及当前节点都要变动
        System.out.println("                     "+stu.getName() + "添加了！！！！ ");        //调试方法哦！
}       //记好了加入后游标指到加入元素的位置，与后面的方法有差异哦

public void delete()throws NoRecordAvailabeException,Exception{          //删除当前位置的元素
        try{
                if(size == 0||cursor==size){             //没有元素或已经到达末端就加载后面的元素
                        util.loadBack();
                }
        }catch(Exception e){}
        if(size==0||cursor==0) throw new NoRecordAvailabeException("当前位置不能删除或没有任何记录！ ");
                                        //加载不成功则抛出异常，上面的逻辑有一些问题！自行调整吧
        if(cursor <= size){                      //一般都是处于这种情况下！删除
                curr.prev.next=curr.next;
                curr.next.prev=curr.prev;
                curr=curr.prev;                  //上面是删除当前元素时指针的变化情况
                size--;
                cursor--;                        //删除当前元素时元素数量、游标的变化情况
        }else{
                throw new Exception("系统有严重错误！ ");
        }
}

public Student next()throws NoRecordAvailabeException,Exception{          //遍列下一个元素
        try{
                if(size==0||cursor >= size){                 //没有元素或者到达最末端加载元素
                        util.loadBack();
                }
        }catch(Exception e ){}
        if(size==0) throw new NoRecordAvailabeException("你的数据库中已经没有记录！ ");
                                                //加载后仍然没有则数据库肯定没有记录，抛出异常
```

```
        if(cursor < size){                                  //移动游标到下一个元素
            curr = curr.next;
            cursor ++;
        }else if(cursor==size){
            throw new NoRecordAvailabeException("已经到达最末端！ ");
        }else if(cursor > size){
            throw new Exception("系统有严重错误！ ");
        }
        return curr.elem ;                                  //返回当前记录的值
    }

    public Student previous()throws Exception{              //和 next 方法类似
        try{
            if(cursor<= 1 ){
                util.loadHeader();
            }
        }catch(Exception e){}
        if(size==0) throw new NoRecordAvailabeException("你的数据库中已经没有记录！ ");
        if(cursor==1) throw new Exception("你已经到达最前端！ ");
        curr = curr.prev;
        cursor--;
        return curr.elem;
    }

    /**public*/private void verbose(){            //自己调试时的一个简单的方法，用于跟踪内存中元素的状态
        System.out.println();System.out.println();
        System.out.println("size =   "+ size + "   position   "+ cursor );
        System.out.println();System.out.println();
    }

    /**public*/private Student removeFirst()throws Exception{          //删除链表的第一个元素，简单吧！
        if(size<=0)  throw  new  NoRecordAvailabeException(" 当 前 位 置 不 能 删 除 或 没 有 任 何 记
录？ ？ ？ ？ ？ ？ ？ ？ ！ ");
        StuNode tmp = header.next;
        header.next.next.prev=header;//order
        header.next=header.next.next;
        cursor--;
        size--;
        if(cursor<=1){
            curr=header;
        }
        return tmp.elem;
    }               //该方法主要用于在内存中记录较多时，从链表头删除元素，放入临时文件中保存

    /**public*/ private Student removeLast()throws Exception{          //删除链表的最后一个元素
        if(size<=0) throw new NoRecordAvailabeException("当前位置不能删除或没有任何记录！ ");
        StuNode tmp = header.prev;
```

```
        header.prev=header.prev.prev;
        header.prev.next=header;
        if(cursor==size){
            curr=header.prev;      //curr=header.prev;
            cursor--;
            size--;
        }else if(cursor<size){
            size--;
        }else{
            throw new Exception("系统严重错误！ ");
        }
        return tmp.elem;
}               //该方法主要用于在内存中记录较多时，从链表尾删除元素，放入文件中保存

/**public*/ private void addBeforeCurrent(Student stu)throws Exception{          //在当前记录前面插入数据
        System.out.println("Enter into addBeforeCurrent !");
        if(size >= capacity){                           //插入时超过容量，则先保存后面的元素
            util.saveBack();
        }
        StuNode tmp = new StuNode(curr.prev,stu,curr);                  //创建一个新的节点
        curr.prev.next=tmp;
        curr.prev=tmp;
        size++      ;
        cursor++;
        System.out.println("size =      "+ size +"      current    "+ curr.elem.getName() );
        System.out.println("Enter off addAfterCurrent !");                  //两个测试方法
}           //调用此方法时，当前元素是不会改变的，注意与 add 方法的差别哦！

/**public*/ private void addAfterCurrent(Student stu)throws Exception{            //和上面的雷同
        System.out.println();
        System.out.println("Enter into addAfterCurrent !");
        if(size >= capacity){
            util.saveHeader();
            System.out.println("元素太多了，头部保存，保存至");
        }
        StuNode tmp = new StuNode(curr,stu,curr.next);
        curr.next.prev=tmp;
        curr.next=tmp;
        size++;
        System.out.println("size =      "+ size    );
        System.out.println("Enter off addAfterCurrent !");
}           //测试用的代码可以去掉

public void save()throws Exception{             //有趣的方法，主要调用内部类的方法完成文件的操作
        util.saveAll();
}
```

```
//下面是一个与文件相关的内部类，主要完成从文件中读数据或向文件中写数据
class DOAFileUtil {                         //有意识对客户隐藏
    LinkedList tmpFile , serFile ;          //当前目录下 ser 目录中.ser 文件和.tmp 文件的集合
                                            //文件的名称是 6 位的，从 100001 到 999999 哦！
                                            //并且文件明是排序的，为什么？看下面的代码就明白用意了
    public DOAFileUtil(){                   //构造器，主要完成创建子目录、得到当前文件的列表
        initFiles();
    }

    private void initFiles(){               //完成一系列初始化工作，同时各添加一个默认的文件
        try{
            tmpFile = searchPersistenceFiles("tmp");        //查找 ser 目录下 tmp 文件，放入列表中
            serFile = searchPersistenceFiles("ser");        //查找 ser 目录下 ser 文件，放入列表中
        }catch(Exception e){
        }finally{
            if(tmpFile == null ){ tmpFile = new LinkedList();}  //基本不会发生此现象
            if(serFile == null ){ serFile = new LinkedList();}
            tmpFile.addFirst(new String("100000.tmp"));     //加一个默认的文件，不必须哦！
            serFile.addFirst(new String("100000.ser"));     //主要明白意图
        }
    }

    public    void loadHeader()throws IOException,NoRecordAvailabeException ,Exception{
                                            //将 tmp 文件中的记录装载入 Cache 中
        checkFile(tmpFile);                 //累赘的检查方法，不必须哦！
        String endFile = (String)tmpFile.removeLast();  //将最近的临时文件名从列表中取出
        File tmp = new File(createSerDir(),endFile);    //得到此临时文件
        ObjectInputStream ois = new ObjectInputStream(new BufferedInputStream(new FileInputStream(tmp)));
                                            //创建对象输入流
        Student stu = null;
        int i=0;
        while(i< STU_NUM /**50**/){         //读取 n 次！
            try{
                stu = (Student)ois.readObject();        //读取每一个记录
            }catch(Exception ee){ }
            if(stu == null) break ;
            addBeforeCurrent(stu);          //将此记录放在当前节点的前面
        }
        ois.close();                        //关闭流
        while(!tmp.delete()){               //删除此临时文件，因为记录已经被读出
            ;
        }                                   //因此必须保证此文件被删除！！！
    }

    public    void loadBack()throws Exception{          //有上面的方法雷同，不讲述了
        checkFile(serFile);
```

```
        String prvFile = (String)serFile.removeLast();
        File ser= new File(createSerDir(),prvFile);
        ObjectInputStream ois = new ObjectInputStream(new BufferedInputStream(new FileInputStream(ser)));
        Student stu =null ;
        int i =0 ;
        while(i< STU_NUM /**50**/){
                i++;
                try{
                        stu = (Student)ois.readObject();
                }catch(Exception e){   }
                if(stu == null ) break ;
                addAfterCurrent(stu);
        }
        ois.close();
        while(!ser.delete()){
                ;
        }
}

private void    checkFile(List l)throws NoRecordAvailabeException{
        if(l.size()==0) throw new NoRecordAvailabeException("你的数据库中没有记录！ ");
}                       //累赘的方法！

public    void saveHeader()throws Exception{            //将头部的 50 个记录保存到临时文件中
        String       file ;//= "100000.tmp";
        if(tmpFile.size()==0){
                file = "100000.tmp";
        }else{
                file =incrFileName(tmpFile);
        }                                               //以上是得到最新的临时文件的名字
        File tmp = new File(createSerDir(),file);
        saveNumOfRecordsFromHeaderToFile(50,tmp);       //将 n 个元素保存到指定文件中
        tmpFile.addLast(tmp.getName());                 //必须将此临时文件的名称加入列表中哦
}

public    void saveBack()throws Exception{              //与上面的方法雷同
        String       file ;
        if(serFile.size()==0){
                file = "100000.ser";
        }else{
                file =incrFileName(serFile);
        }
        File ser = new File(createSerDir(),file);
        saveNumOfRecordsFromLastToFile(50,ser);
        serFile.addLast(ser.getName())   ;
}
```

```
    private    void saveNumOfRecordsFromLastToFile(int i,File file)throws Exception{
        ObjectOutputStream      oos     =     new     ObjectOutputStream(new     BufferedOutputStream(new
FileOutputStream(file)));
        //创建对象的输出字节流
        if(i<=0) return ;              //没有对象就没有必要做了
        for(int j = 0 ; j<i ; j++){
            oos.writeObject(removeLast());              //将 Cache 中最后的元素依次放入文件中
        }
        oos.flush();                                    //带缓冲的流在关闭前肯定要清空了
        oos.close();                                    //关闭流，完成记录的写入
    }

    private    void saveNumOfRecordsFromHeaderToFile(int i,File file)throws Exception{     //同上
        ObjectOutputStream      oos     =     new     ObjectOutputStream(new     BufferedOutputStream(new
FileOutputStream(file)));
        if(i<=0) return ;
        for(int j = 0 ; j<i ; j++){
            oos.writeObject(removeFirst());   //removeFirst//not true
        }
        oos.flush();
        oos.close();
    }

    public    void saveLastRecord(int num)throws Exception{    //冗余方法
        String      file ;//= "100000.tmp";
        if(tmpFile.size()==0){
            file = "100000.tmp";
        }else{
            file =incrFileName(serFile);
        }
        File ser = new File(createSerDir(),file);
        saveNumOfRecordsFromLastToFile(num,ser);
        serFile.addLast(ser.getName())   ;
    }

    private    String incrFileName(LinkedList fileNames)throws Exception {
                    //根据列表中元素的名称确定下一个临时或者 ser 文件的名称
        if(fileNames.size()!=0){}                                   //本程序中不会发生
        String max = (String)fileNames.getLast();//removeLast      //得到最后一个文件名
        StringBuffer sb = new StringBuffer();                       //用于保存要生成文件的各个部分
        String oldPrefix = max.substring(0,max.indexOf("."));       //查找“.”的位置
        String ext = max.substring(max.indexOf("."));               //文件后缀
        int i = Integer.parseInt(oldPrefix);                        //将点前面的部分变为 int
        i++;                                                        //
        Integer pre = new Integer(i);                               //最新产生的文件名
        sb.append(pre.toString()).append(ext);                      //
        return sb.toString();                                       //返回此文件名
```

```
}

private    File createSerDir()throws Exception{    //一个程序中使用的创建 ser 目录的方法，使用多次
        File curDir = new File(".");                //当前目录
        File parent = curDir.getCanonicalFile();    //目录的全部路径
        File f = new File(parent,"Ser");            //当前目录下的 ser 目录
        if(!f.exists()){                            //该目录不存在则创建此目录，必须创建成功
            while(!f.mkdir()){
                ;
            }
        }
        //System.out.println(f);
        return f;                                   //返回此文件
}

public void renameTmpToSer()throws Exception{
            //一个在 save 方法中使用的方法，用于程序结束时的善后工作
        File dir = createSerDir();          //该方法在 ser 目录已经存在时，就是一个得到目录的方法
        while(tmpFile.size()>1){            //将文件列表中所有的临时文件按次序转变为 ser 文件
            String renameTo =    incrFileName(serFile);
            //tmp 文件改名为 ser 文件，所以 ser 中要先创建一个将要加入的 ser 文件的文件名
            String tmp = (String)tmpFile.removeLast();      //同时 tmp 文件要被删除
            File file = new File(dir,tmp);                  //得到此临时文件
            file.renameTo(new File(dir,renameTo));          //将临时文件改名为 ser 文件
            serFile.addLast(renameTo);                      //必须将此最新的 ser 文件放到末端
        }
}           //次序很重要，主要和你输入记录的次序有关！否则可能面目全非哦

        public void saveAll()throws Exception{
            //在程序关闭后，将内存中所有的记录保存到文件中，一遍以后应用
            File dir = createSerDir();          //得到此目录
            while(size>0){                      //当内存中有记录就保存记录，知道没有记录
                int i = 0 ;
                String      file ;
                if(serFile.size()==0){
                    file = "100000.ser";
                }else{
                    file =incrFileName(serFile);
                }                               //得到最后这个要加入文件的文件名
                File ser = new File(dir,file);      //创建这个文件
                ObjectOutputStream  oos  =  new  ObjectOutputStream(new  BufferedOutputStream(new
FileOutputStream(ser)));
                while(i<STU_NUM&&size>0){           //保存 50 个记录
                    oos.writeObject(removeLast());  //将最后的记录依次放入 ser 文件中
                    i++;            //已经保存的数目，主要控制每个文件中不能多余 50 个记录
                }
                oos.flush();            //每 50 个保存完整，情况缓存
```

```
                oos.close();            //关闭流
                serFile.addLast(ser.getName())  ;     //每增加一个 ser 文件，必须放入列表中
            }
        }

    public LinkedList searchPersistenceFiles(final String fileExtension)throws Exception{
        //根据文件的后缀名将此后缀的文件放入列表中
        File ser = createSerDir();
        //下面是一个返回 ser 目录中指定后缀，且符合名称约定的文件名的数组
        //主要是一个内部类，关于内部类请关注博客中其它文章
        // FilenameFilter 在设计模式中称为“可换滤芯”模式
        String[] serSiles = ser.list(
            new FilenameFilter(){
                public boolean accept(File dir, String name){
                    if(!name.endsWith("."+fileExtension)) return false ;
                    boolean flag = false ;
                    try{
                        int i = Integer.parseInt(name.substring(0,name.indexOf(".")));
                        if(i>100000 &&i<999999 ){           //有上面的经验可以明白吧！
                            flag = true ;
                        }
                    }catch(Exception e){  }
                    return flag ;
                }
            }
        );
        List l = Arrays.asList(serSiles);                   //通过 Arrays 工具类将数组转变为 List
        Collections.sort(l);                                //将名称排序
        LinkedList list = new LinkedList(l);                //根据其中的内容创建一个链表
                    //使用链表是因为我们要从最前端和最后端增加或者删除元素
        return list;                                        //返回此链表
                    //短短几句使用了 Java Collections Framework 中的不少内容，有意思吧！
                    //是否你对于这些数据结构都很了解呢！？仔细研究吧！
    }
  }
}
```

很累吧！为了一个简单的想法－－不要一次将所有的记录加入内存，我们设计了这个近十页的数据结构！它完成从文件系统将记录加载到内存，将内存中的记录保存到文件中，很有趣吧！

简单的数据结构就是 10 页，假如一个复杂的程序呢？至少几千页吧！如何设计他们、组织他们是需要理论基础的！下篇会讲述本程序的结构！