第 5 章　输入/输出

5.1　知 识 简 介

输入/输出(I/O)是软件与外部世界交互的关键接口，I/O 对软件系统的主要作用有：①实现软件系统与用户交互；②提供数据存储和检索；③实现与外部设备通信；④实现网络通信；⑤实现数据格式化和解析。

Java 的 I/O 基于流(Stream)技术。流根据其字节多少分为字节流和字符流。字节流以字节(byte)为单位进行 I/O 操作，适用于处理二进制数据，如图像、音频、视频等，以及文本数据。主要字节流类包括 InputStream 和 OutputStream。例如，FileInputStream 用于从文件读取字节数据，FileOutputStream 用于向文件写入字节数据。字符流以字符(char)为单位进行 I/O 操作，适用于处理文本数据，通常涉及字符编码(如 UTF-8、ISO-8859-1 等)的转换。主要字符流类包括 Reader 和 Writer。例如，FileReader 用于从文件读取字符数据，FileWriter 用于向文件写入字符数据。字符流适用于处理文本文件，如配置文件、日志文件、文档文件等，以及与用户进行文本交互的场景。

流根据其在 I/O 流层次结构中的位置和作用分为节点流和处理流。节点流(Node Streams)是直接与底层数据源(如文件、内存、网络连接)连接的流，也称底层流(Low-Level Streams)。主要节点流包括 InputStream 和 OutputStream(用于字节数据)以及 FileReader 和 FileWriter(用于字符数据)。处理流(Filter Streams)是构建在节点流之上的流，也称包装流(Wrapper Streams)。处理流用于缓冲、过滤、转换数据等高级操作，通常提供更高层次的 API 来简化 I/O 操作。主要处理流包括 BufferedInputStream 和 BufferedOutputStream(用于缓冲字节流)以及 InputStreamReader 和 OutputStreamWriter(用于字符编码转换)等。

图 5-1 显示字节输出流的主要类结构。其中 ByteArrayOutputStream、FileOutputStream、PipedOutputStream 是节点流，FilterOutputStream 子类以及 ObjectOutputStream 是处理流。

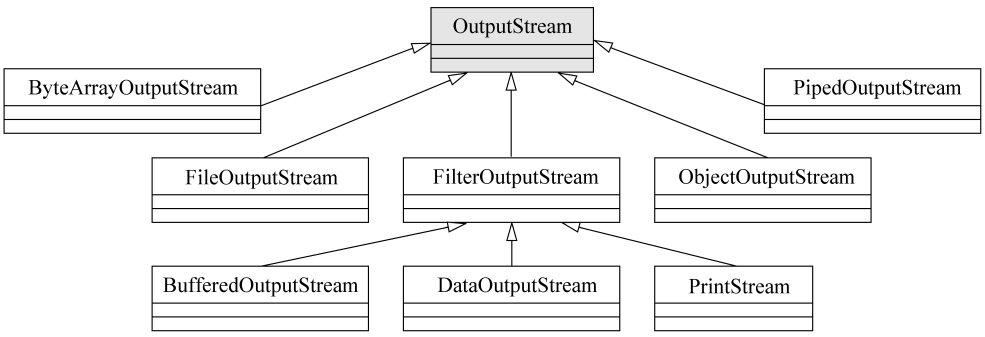

图 5-1　字节输出流的主要类结构

图 5-2 显示字节输入流的主要类结构。ByteArrayInputStream、FileInputStream、

PipedInputStream 等是节点流，FilterOutputStream 子类以及 ObjectOutputStream、StringBufferInputStream 等是处理流。

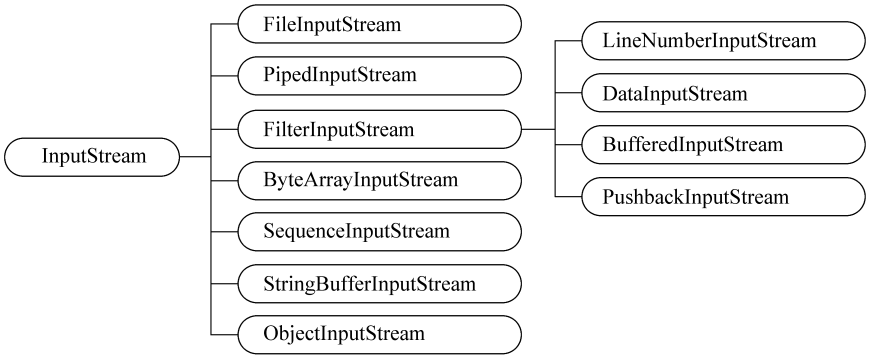

图 5-2　字节输入流的主要类结构

图 5-3 显示字符输出流的主要类结构。CharArrayWriter、FileWriter、PipedWriter 等是节点流，FilterWriter 子类以及 StringWriter、BufferedWriter、PrintWriter 等是处理流。

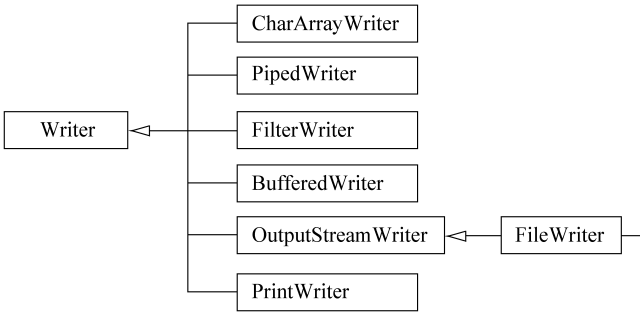

图 5-3　字符输出流的主要类结构

图 5-4 显示字符输入流的主要类结构。CharArrayReader、FileReader、PipedReader 等是节点流，FilterReader 子类以及 StringReader、BufferedReader 等是处理流。

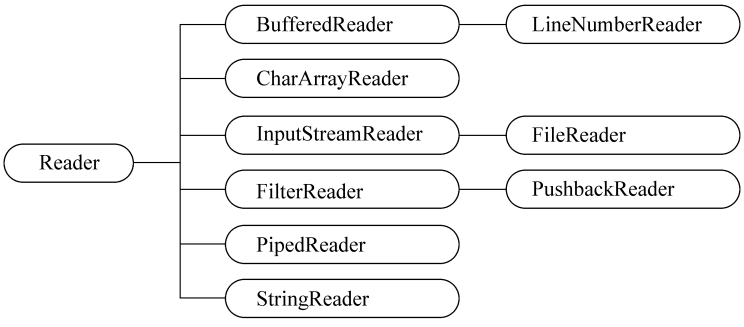

图 5-4　字符输入流的主要类结构

对象序列化指把 Java 对象转换为字节流的过程，反序列化是将字节流转换到对象的过程。通俗来说，对象序列化允许将对象的状态保存到文件、数据库或通过网络传输，并在需要时重新构建对象。通过序列化与反序列化，可以实现不同系统之间的数据传输或持久化存储。一个对象要能够被序列化，前提条件是该对象的类实现 java.io.Serializable 接口。

Java 提供了 ObjectOutputStream 和 ObjectInputStream 两个类，用于将对象序列化为字节流或将字节流反序列化为对象。

5.2 实践目的

通过项目实践，读者熟悉 Java 的 I/O 框架结构，了解各种类的作用，掌握读取数据、写入数据、数据转换、数据持久化存储和恢复等知识，加深读者对数据在计算机系统中流动过程的理解，培养读者针对实际问题建立 I/O 处理模型，选择合适的 I/O 类，设计简单应用系统的能力。

5.3 实践范例

5.3.1 范例 1 文件处理工具类问题

1. 范例描述

使用 Java 的 I/O 框架定义文件处理工具类，包括复制文件、统计文本文件中某个词出现的次数、查找替换文本文件内容、把 Excel 文件中的每个表单转换成一个文本文件。该问题的具体要求如下。

(1) 复制文件使用缓冲流和 NIO 两种方式。

(2) 使用缓冲流统计文本文件中某个词出现的次数。

(3) 查找替换文本文件内容时，需要使用指定字符串替换所有匹配字符串。

(4) 把 Excel 文件中的每个表单转换成一个文本文件，Excel 的每行生成文本文件的一行，Excel 的每列在文本文件中用制表符分隔。

2. 范例分析

文件处理工具类中应对不同要求需要使用不同流，具体如下。

(1) 复制文件使用缓冲流和 NIO 两种方式。复制文件需要使用 FileInputSteam 和 FileOutputStream，为了提高效率可以使用缓冲流 BufferedInputStream 和 BufferedOutputStream，使用 NIO 复制文件使用 FileChannel 类。

(2) 使用缓冲流统计文本文件中某个词出现的次数。文本文件的读取使用 FileReader，对应的缓冲流为 BufferedReader。

(3) 查找替换文本文件内容时，使用指定字符串替换所有匹配字符串。为了实现查找替换操作，首先把文本文件读取到字符串 StringBuilder，然后使用 String 的 replaceAll()方法替换，最后把替换结果写入文本文件。

(4) 把 Excel 文件中的每个表单转换成一个文本文件，Excel 的每行生成文本文件的一行，Excel 的每列在文本文件中用制表符分隔。①创建了一个名为 outputDirectory 的目录用于存放生成的文本文件。②遍历每个工作表时为每个工作表创建一个单独的文本文件，并以工作表的名称命名这些文件。③使用 Apache POI 读取 Excel 文件，遍历工作表、行和单元格，将单元格的内容写入文本文件，分隔符使用制表符(\t)，每行数据用换行符(\n)分隔。

3. 范例代码

```
import java.io.*;
import java.nio.channels.FileChannel;
import java.nio.file.*;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;
import org.apache.poi.ss.usermodel.*;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import java.util.Iterator;
//文件实用工具类
public class FileUtil {
    //复制文件(使用缓冲流)
    public static void copyFile(String sourcePath, String destinationPath)
            throws IOException {
        try (BufferedInputStream inputStream = new BufferedInputStream(
                new FileInputStream(sourcePath));
                BufferedOutputStream outputStream = new BufferedOutputStream(
                        new FileOutputStream(destinationPath))) {

            byte[] buffer = new byte[1024];
            int bytesRead;
            while ((bytesRead = inputStream.read(buffer)) != -1) {
                outputStream.write(buffer, 0, bytesRead);
            }
        }
    }
    //使用 NIO 复制文件
    public static void copyFileWithNIO(String sourceFilePath,
            String destinationFilePath) throws IOException {
        Path sourcePath = FileSystems.getDefault().getPath(sourceFilePath);
        ;
        Path destinationPath = FileSystems.getDefault().getPath(
                destinationFilePath);
        try (FileChannel sourceChannel = FileChannel.open(sourcePath,
                StandardOpenOption.READ);
                FileChannel destinationChannel = FileChannel.open(
                        destinationPath, StandardOpenOption.CREATE,
                        StandardOpenOption.WRITE)) {
            long transferredBytes = 0;
            long fileSize = sourceChannel.size();
            while (transferredBytes < fileSize) {
                long bytesTransferred = sourceChannel.transferTo(
                        transferredBytes, fileSize - transferredBytes,
                        destinationChannel);
                transferredBytes += bytesTransferred;
            }
            System.out.println("文件复制完成。");
        } catch (IOException e) {
```

```
            e.printStackTrace();
            throw e;                          //将异常重新抛出以便上游处理
        }
    }
    //读取文本文件
     public static List < String > readTextFile ( String filePath ) throws
IOException {
        List<String> lines = new ArrayList<>();
        try (BufferedReader reader = new BufferedReader(
                new FileReader(filePath))) {
            String line;
            while ((line = reader.readLine()) != null) {
                lines.add(line);
            }
        }
        return lines;
    }
    //搜索文本文件(不换行)
    public static List<String> searchInTextFile(String filePath,
            String searchQuery) throws IOException {
        List<String> matchingLines = new ArrayList<>();
        StringBuilder fileContent = new StringBuilder();
        try (BufferedReader reader = new BufferedReader(
                new FileReader(filePath))) {
            String line;
            while ((line = reader.readLine()) != null) {
                fileContent.append(line);  //将每行文本拼接到字符串中,不包括换行符
            }
        }
        String fileContentStr = fileContent.toString();
        int index = 0;
        while ((index = fileContentStr.indexOf(searchQuery, index)) != -1) {
            int lineStart = Math
                    .max(0, fileContentStr.lastIndexOf("\n", index));
            int lineEnd = Math.min(
                    fileContentStr.indexOf("\n", index + searchQuery.length()),
                    fileContentStr.length());
            matchingLines.add(fileContentStr.substring(lineStart, lineEnd));
            index += searchQuery.length();
        }
        return matchingLines;
    }
    //查找替换文本文件中的内容(不换行)
    public static void replaceInTextFile(String filePath, String searchQuery,
            String replacement) throws IOException {
        StringBuilder fileContent = new StringBuilder();

        try (BufferedReader reader = new BufferedReader(
                new FileReader(filePath))) {
            String line;
```

```
            while ((line = reader.readLine()) != null) {
                fileContent.append(line);  //将每行文本拼接到字符串中,不包括换行符
            }
        }
        String fileContentStr = fileContent.toString();
        String updatedContent = fileContentStr
                .replace(searchQuery, replacement);

        try (BufferedWriter writer = new BufferedWriter(
                new FileWriter(filePath))) {
            writer.write(updatedContent);
        }
    }
    //删除文件
    public static boolean deleteFile(String filePath) {
        File file = new File(filePath);
        return file.delete();
    }
    public static void convertExcelToText(String excelFilePath,
            String outputDirectory) throws IOException {
        FileInputStream excelFile = new FileInputStream(new File(excelFilePath));
        Workbook workbook = new XSSFWorkbook(excelFile);
        //创建输出目录
        File outputDir = new File(outputDirectory);
        if (!outputDir.exists()) {
            outputDir.mkdirs();
        }
        //遍历每个工作表
        for (Sheet sheet : workbook) {
            String sheetName = sheet.getSheetName();
            String textFilePath = outputDirectory + File.separator + sheetName+
".txt";
            //打开文本文件以进行写入
            FileWriter writer = new FileWriter(textFilePath);
            BufferedWriter bufferedWriter = new BufferedWriter(writer);
            Iterator<Row> rowIterator = sheet.iterator();
            //遍历每一行
            while (rowIterator.hasNext()) {
                Row row = rowIterator.next();
                //遍历每个单元格并将其写入文本文件
                Iterator<Cell> cellIterator = row.iterator();
                while (cellIterator.hasNext()) {
                    Cell cell = cellIterator.next();
                    String cellValue = getCellValueAsString(cell);
                    bufferedWriter.write(cellValue);
                    bufferedWriter.write("\t");  //以制表符分隔单元格
                }
                bufferedWriter.newLine();        //换行以分隔行
            }
            //关闭文件流
```

```
            bufferedWriter.close();
            writer.close();
        }
        System.out.println("Excel 文件的每个工作表已成功转换为文本文件并保存在目
录:" + outputDirectory);
    }
    //辅助方法:将单元格内容转换为字符串
    private static String getCellValueAsString(Cell cell) {
        String cellValue = "";
        if (cell != null) {
            switch (cell.getCellType()) {
            case STRING:
                cellValue = cell.getStringCellValue();
                break;
            case NUMERIC:
                cellValue = String.valueOf(cell.getNumericCellValue());
                break;
            case BOOLEAN:
                cellValue = String.valueOf(cell.getBooleanCellValue());
                break;
            case BLANK:
                cellValue = "";
                break;
            default:
                cellValue = "";
            }
        }
        return cellValue;
    }
}
```

4. 运行结果

文件处理工具类问题的测试代码如下,测试前需要在相关目录下建立文件,或已经建立了保存文件的目录。

```
import java.io.IOException;
import java.util.List;
public class FileUtilMain {
    public static void main(String[] args) {
        try {
            //使用 NIO 复制文件
            FileUtil.copyFileWithNIO("d:\\人工智能发展方案.zip","d:\\人工智能发
展方案-x.zip");
            //使用传统方式复制文件
            FileUtil.copyFile("d:\\人工智能发展方案.zip","d:\\人工智能发展方案-x.
zip");
            //把 Excel 文件转换成文本文件
            FileUtil.convertExcelToText("C:\\poi-班级学生名单.xlsx", "C:\\poi-
班级学生名单.txt");
```

```
            //读取文本文件的数据
            List<String> listTxt=FileUtil. readTextFile("d:\\pom.xml");
            for(String txt:listTxt)
                System.out.println(txt);
            FileUtil.replaceInTextFile("d:\\pom.xml","dep","--XXXXX--");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

5.3.2 范例 2　科技成果管理系统问题

1. 范例描述

科技成果不仅代表企业(组织)的创新能力和技术水平,也是企业(组织)的核心资产和竞争优势,设计开发科技成果管理系统对于依赖创新和知识产权来推动业务发展的企业(组织)至关重要。科技成果管理系统的主要作用是保护知识产权、促进技术转移和商业化、加强合作伙伴关系、提高创新效率、支持战略决策和提升组织的声誉和形象。通过有效地管理科技成果,企业(组织)可以更好地实现其创新战略,提高竞争力和可持续发展能力。

科技成果包括论文、发明专利、软件著作权和外观设计专利等。设计科技成果管理系统要求如下:保存科技成果信息、统计各种类型成果数量、根据关键字查找科技成果、打印科技成果信息。该问题的具体要求如下:

(1) 增加成果信息;

(2) 删除成果信息;

(3) 显示各种类型科技成果数量;

(4) 根据关键字查找科技成果,并显示成果信息;

(5) 按类型显示科技成果信息。

2. 范例分析

科技成果管理系统实现科技成果管理,科技成果包括论文、发明专利、软件著作权和外观设计专利等,需要设计“科技成果”父类,父类中有成果类型,子类包括发明专利、软件著作权和外观设计专利,如图 5-5 所示。

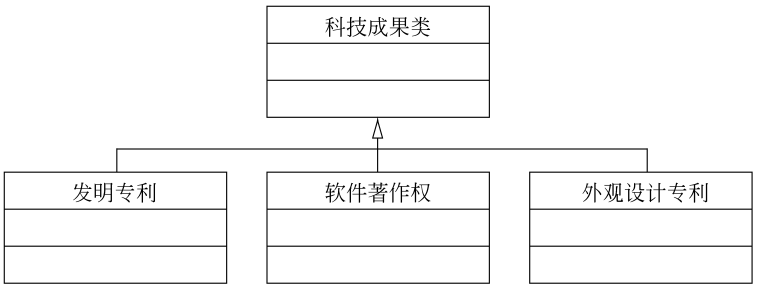

图 5-5　科技成果管理系统问题的 UML 类结构

科技成果管理系统的每个功能的解决要点如下。

(1) 增加成果信息。科技成果保存在文件中便于存储和传输,需要使用 I/O 框架的序列化和反序列化技术,涉及类 ObjectOutStream 和 ObjectInputStream。先读取文件中已有对象到集合,然后将新的对象添加到集合尾部,最后将集合中的所有对象重新写入文件。

(2) 根据成果名删除成果信息。使用反序列化技术,读取成果信息并保存在集合,在集合中查找需要删除的成果并删除,然后通过序列化技术把集合中的成果信息写入文件。

(3) 显示各种类型科技成果数量。使用反序列化技术从文件中读取成果对象,根据成果类型分别统计数量,使用 Map<类型,数量>记录各种类型成果信息。

(4) 根据关键字查找科技成果。使用反序列化技术从文件中读取成果信息,在成果所有内容中查找符合关键字的成果,把查找结果保存在集合中。

(5) 按类型显示科技成果信息。通过反序列化技术,把保存在文件中的科技成果信息提取到集合中,使用 Collections 类的 sort()方法对集合进行排序,最后显示排序后的成果信息。需要定义成果类型为枚举类型,为每个常量赋值,并能获得该常量的赋值,在成果类中实现比较接口 Comparable,根据成果类型进行比较。

3. 范例代码

科技成果管理系统的类结构如图 5-6 所示,它包含 7 个类,AchievementMain 是测试类,TechnologicalAchievement 是抽象类,实现 Serializable 和 Comparable 两个接口,AchievementType 成果类型是枚举,DesignPatent(外观设计)、InventionPatent(发明专利)、SoftwareCopyright (软件著作权)类继承 TechnologicalAchievement。

图 5-6　科技成果管理系统的类结构

```
import java.io.Serializable;
//父类:科技成果
public abstract class TechnologicalAchievement implements Serializable,
Comparable<TechnologicalAchievement> {
    private static final long serialVersionUID = 1L;
    //private static final long serialVersionUID = 1L;
    private String name;                        //成果名称
    private String creator;                     //成果创造者
    private String description;                 //成果描述
    private AchievementType type;               //成果类型

    public TechnologicalAchievement() {
        super();
    }
    //构造方法
```

```
    public TechnologicalAchievement(String name, String creator, String
description, AchievementType type) {
        this.name = name;
        this.creator = creator;
        this.description = description;
        this.type = type;
    }
    //省略 Getter 和 Setter 方法
    //方法:返回科技成果的所有信息
    public String getInfo() {
        return "名称:" + this.name + ",创造者:" + this.creator + ",描述:" + this.
description + ",类型:" + this.type+this.getOther();
    }
    //抽象方法,具体实现由子类提供
    public abstract void displayDetail();
    public abstract String getOther();
    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((name == null) ? 0 : name.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        TechnologicalAchievement other = (TechnologicalAchievement) obj;
        if (name == null) {
            if (other.name != null)
                return false;
        } else if (!name.equals(other.name))
            return false;
        return true;
    }
    @Override
    public int compareTo(TechnologicalAchievement other) {
        if(this.getType().getNumber()>other.getType().getNumber())
            return 1;
        else
            return -1;
    }
    @Override
    public String toString() {
        return "TechnologicalAchievement{" +
```