福建師範大學
FUJIAN NORMAL UNIVERSITY

Java语言

学　　期：2025-2026-1

教　　师：赵珊珊

第9章I/O流

第9章I/O流

- 9.1 I/O流概述
- 9.2 字节流
- 9.3字符流
- 9.4File类
- 9.5RandomAccessFile类

9.1 I/O流概述

- I/O（Input/Output）流，即输入输出流，是Java中实现输入输出的继承，它可以方便地实现数据的输入输出操作。
- Java中的“流”都位于java.io包中，按照操作数据的不同，可以分为字节流和字符流，按照数据传输方向的不同又可分为输入流和输出流，程序从输入流中读取数据，向输出流中写入数据。

I/O流分类	字节流	字符流
输入流	InputStream	Reader
输出流	OutputStream	Writer

9.1 I/O流概述

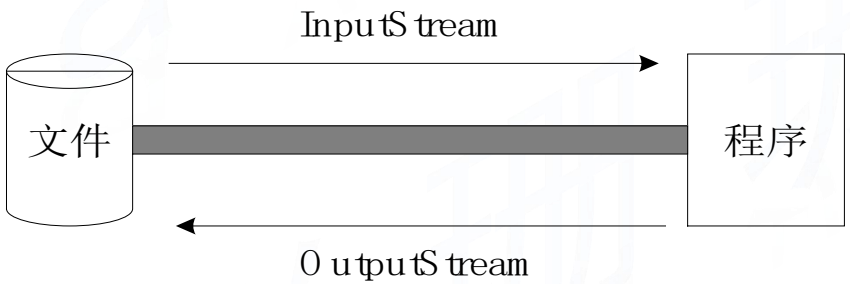
- 1.字节流和字符流
- 根据流操作的数据单位的不同，可以分为字节流和字符流。字节流以字节为单位进行数据的读写，每次读写一个或多个字节数据；字符流以字符为单位进行数据的读写，每次读写一个或多个字符数据。
- 2.输入流和输出流
- 根据流传输方向的不同，又可分为输入流和输出流。其中输入流只能从流中读取数据，而不能向其写入数据；输出流只能向流中写入数据，而不能从中读取数据。

9.2 字节流

- 在计算机中，无论是文本、图片、音频还是视频，所有的文件都是以二进制(字节)形式存在，I/O流中针对字节的输入输出提供了一系列的流，统称为字节流。字节流是程序中最常用的流，根据数据的传输方向可将其分为字节输入流和字节输出流。

9.2 字节流

- 在JDK中，提供了两个抽象类InputStream和OutputStream，它们是字节流的顶级父类，所有的字节输入流继承自InputStream，所有的字节输出流都继承自OutputStream。



9.2 字节流

- 9.2.1字节流读写文件
- 在操作文件时，最常见的就是从文件中读取数据并将数据写入文件。针对文件的读写，JDK专门提供了两个类，分别是FileInputStream和FileOutputStream，其中FileInputStream是InputStream的子类，它是操作文件的字节输入流，专门用于读取文件中的数据。

9.2 字节流

- 9.2.1字节流读写文件
- 与FileInputStream对应的是FileOutputStream。FileOutputStream是OutputStream的子类，它是操作文件的字节输出流，专门用于把数据写入文件。

inputstream example

```
import java.io.FileInputStream;
import java.io.IOException;
public class FileIOStreamExample {
    public static void main(String[] args) {
        // 定义缓冲区大小

        FileInputStream fis=null;
        try{
             fis = new FileInputStream("d:/in.txt");
             int n = 32;
             byte buffer[]=new byte[n];
            // 读取数据到缓冲区，并确保读取长度不超过缓冲区大小
            while ((fis.read(buffer, 0, n) != -1) && (n>0)) {
                System.out.print(new String(buffer));
            }
        } catch (IOException e) {
            System.out.println(e);
        }finally {
                y {
                    fis.close();
                }catch(IOException e) {
                    e.printStackTrace();
                }
            }
    }
}
```

9.2 字节流

- 9.2.1字节流读写文件

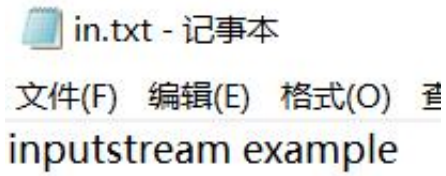
```
import java.io.*;
public class FileIOStreamExample {
    public static void main(String[] args) throws IOException{
        System.out.println("输入要保存的内容：");
        int count, n=512;
        byte buffer[]=new byte[n];
        count = System.in.read(buffer);
        FileOutputStream fos = new FileOutputStream("d:/out.txt");
        fos.write(buffer,0,count);
        System.out.println("已保存到out.txt");
        fos.close();
    }
}
```

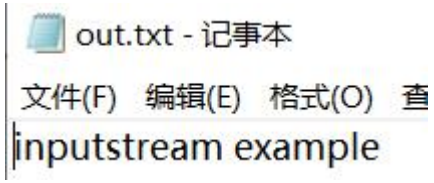
```
输入要保存的内容：
output
已保存到out.txt
```

9.2 字节流

- 9.2.2文件的拷贝
- 在应用程序中，I/O流通常都是成对出现的，即输入流和输出流一起使用。

```
import java.io.*;
public class FileIOStreamExample {
    public static void main(String[] args) throws IOException{
        FileInputStream fis = new FileInputStream("d:/in.txt");
        FileOutputStream fos = new FileOutputStream("d:/out.txt");
        int c;
        while((c=fis.read())!=-1) {
            fos.write(c);
        }
        fis.close();
        fos.close();
    }
}
```





9.2 字节流

- 9.2.3字节流的缓冲区
- 前面讲解了如何复制文件，但复制的方式是一个字节一个字节地复制，频繁操作文件，效率非常低，利用字节流的缓冲区可以解决这一问题，提高效率。

```
import java.io.*;
public class FileIOStreamExample {
    public static void main(String[] args) throws IOException{
        FileInputStream fis = new FileInputStream("d:/in.txt");
        FileOutputStream fos = new FileOutputStream("d:/out.txt");
        byte[] b =new byte[512];
        int len;
        long begin = System.currentTimeMillis();
        while((len=fis.read())!=-1) {
            fos.write(b,0,len);
        }
        long end = System.currentTimeMillis();
        System.out.println("复制文件耗时："+(end-begin)+"毫秒");
        fos.close();
        fis.close();
    }
}
```

9.2 字节流

- 9.2.4字节缓冲流
- 在IO包中提供两个带缓冲的字节流，分别是BufferedInputStream和BufferdOutputStream。它们的构造方法中分别接收InputStream和OutputStream类型的参数作为被包装对象，在读写数据时提供缓冲功能。

```
import java.io.*;
public class FileIOStreamExample {
    public static void main(String[] args) throws IOException{
        FileInputStream fis = new FileInputStream("d:/in.txt");
        FileOutputStream fos = new FileOutputStream("d:/out.txt");
        BufferedInputStream bis = new BufferedInputStream(fis);
        BufferedOutputStream bos = new BufferedOutputStream(fos);
        int c;
        while((c=bis.read())!=-1) {
            bos.write(c);
        }
        bos.close();
        bis.close();
    }
}
```

9.2.4 字节缓冲流

9.3字符流

- Java还提供了字符流，用于操作字符。与字节流相似，字符流也有两个抽象基类，分别是Reader和Writer，Reader是字符输入流，用于从目标文件读取字符；Writer是字符输出流，用于向目标文件写入字符。
- 图9-5和图9-6表示了Reader和Writer的子类。

9.3字符流

- 9.3.1字符流读写文件
- 在程序开发中，经常需要对文本文件的内容进行读取，如果想从文件中直接读取字符便可以使用字符输入流FileReader，通过此流可以从关联的文件中读取一个或一组字符。
- 如果要向文件中写入字符就需要使用FileWriter类。FileWriter是Writer的一个子类，专门用于将字符写入文件。

```
import java.io.*;
public class FileRWExample {
    public static void main(String[] args) throws Exception{
        File f =new File("d:/in.txt");
        FileReader fr =new FileReader(f);
        int len;
        while((len=fr.read())!=-1) {
            System.out.print((char)len);
        }
        fr.close();
    }
}
```

in.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V)

reader and writer example

```
reader and writer example
```

```
import java.io.*;
public class FileRWExample {
    public static void main(String[] args) throws Exception{
        File f =new File("d:/out.txt");
        FileWriter fw =new FileWriter(f);//追加内容(f,true)
        fw.write("FileWriter exmaple");
        System.out.println("已保存到out.txt");
        fw.close();
    }
}
```

out.txt - 记事本

文件(F) 编辑(E) 格式(O)

FileWriter exmaple

9.3字符流

- 9.3.2字符流的缓冲区
- 通过字符流的形式完成了对文件内容的读写操作，是一个一个字符来完成的，这样还是需要频繁的读写文件，效率十分低，这里也可以使用字符流缓冲区进行读写操作，来提高执行的效率。

```
import java.io.*;
public class FileRWExample {
    public static void main(String[] args) throws Exception{
        FileReader fr =new FileReader("d:/in.txt");
        FileWriter fw =new FileWriter("d:/out.txt");//追加内容(f,true)
        int len = 0;
        char buf[] =new char[1024];
        while((len=fr.read(buf))!=-1) {
            fw.write(buf,0,len);
        }
        fr.close();
        fw.close();
    }
}
```

9.3字符流

- 9.3.3字符缓冲流
- 在字节流中提供了带缓冲区功能的字节缓冲流，同样字符流中也提供了带缓冲区的流，分别是BufferedReader类和BufferedWriter类，其中BufferedReader类用于对字符输入流进行包装，BufferedWriter类用于对字符输出流进行包装，包装后会提高字符流的读写效率。

```
import java.io.*;
public class FileRWExample {
    public static void main(String[] args) throws IOException{
        FileReader fr =new FileReader("d:/in.txt");
        FileWriter fw =new FileWriter("d:/out.txt");
        BufferedReader br = new BufferedReader(fr);
        BufferedWriter bw = new BufferedWriter(fw);
        String str;
        while((str=br.readLine())!=null) {
            bw.write(str);
            bw.newLine();
        }
        bw.close();
        br.close();
    }
}
```

9.3字符流

- 9.3.4转换流
- JDK中提供了可以将字节流转换为字符流的两个类，分别是InputStreamReader类和OutputStreamWriter类，它们被称为转换流，其中OutputStreamWriter类可以将一个字节输出流转换成字符输出流，而InputStreamReade类可以将一个字节输入流转换成字符输入流，转换流的出现方便了对文件的读写，它在字符流与字节流之间架起了一座桥梁，使原本没有关联的两种流操作能够进行转化，提高了程序的灵活性。

```
import java.io.*;
public class ConvertExample {
    public static void main(String[] args) throws Exception{
        FileInputStream fis =new FileInputStream("d:/in.txt");
        InputStreamReader isr =new InputStreamReader(fis);
        FileOutputStream fos =new FileOutputStream("d:/in.txt");
        OutputStreamWriter osw =new OutputStreamWriter(fos);
        int c;
        while((c=isr.read())!=-1)
            osw.write(c);
        osw.close();
        isr.close();
    }
}
```

9.4File类

- 通过I/O流可以对文件的内容进行读写操作，但对文件本身进行的一些常规操作是无法通过I/O流来实现的，例如创建一个文件、删除或者重命名某个文件、判断硬盘上某个文件是否存在等。针对文件的这类操作，JDK中提供了一个File类，该类封装了一个路径，并提供了一系列的方法用于操作该路径所指向的文件。

9.4File类

- 1.File类的常用方法
- 2.遍历目录下的文件
- 3.文件过滤
- 4.删除文件及目录

9.4File类

• 1.File类的常用方法

- File 类的三个构造方法
- public File(String pathname)

- 功能描述：通过给定的路径名字符串创建一个新的 File 实例。这个路径可以是绝对路径（如 "C:/docs/test.txt"），也可以是相对路径（如 "config/settings.ini"）。该路径指向的文件或目录不一定实际存在。

- public File(String parent, String child)

- 功能描述：根据父路径字符串和子路径字符串创建一个新的 File 实例。系统会自动将父路径和子路径拼接成完整路径。例如，parent为 "C:/docs"，child为 "report.txt"，则最终路径为 "C:/docs/report.txt"。

- public File(File parent, String child)

- 功能描述：根据一个父 File 对象和子路径字符串创建一个新的 File 实例。这在父路径本身就是一个 File 对象时非常有用，例如先创建一个代表目录的 File 对象，再基于它创建文件的 File 对象。

9.4File类

- 1.File类的常用方法

方法声明	功能描述
`public boolean exists()`	判断此 File 对象所表示的文件或目录在磁盘上是否**实际存在**
`public boolean isFile()`	判断此 File 对象是否代表一个**文件**（并且该文件存在）
`public boolean isDirectory()`	判断此 File 对象是否代表一个**目录**（并且该目录存在）
`public String getName()`	返回由此 File 对象表示的文件或目录的**名称**（不包含父路径信息）
`public String getPath()`	返回创建此 File 对象时使用的**路径字符串**（可能是相对或绝对路径）
`public String getAbsolutePath()`	返回此 File 对象所表示的文件或目录的**绝对路径字符串**
`public long length()`	返回由此 File 对象表示的**文件的长度（字节数）**。如果对象是目录或文件不存在，则返回值是**0**（在某些系统上目录返回值未定义）
`public long lastModified()`	返回此 File 对象表示的文件**最后被修改的时间**，返回值是自纪元（1970年1月1日 00:00:00 GMT）以来的毫秒数
`public boolean createNewFile()`	当且仅当具有该名称的文件**尚不存在**时，**创建一个新的空文件。**创建成功返回 `true`，文件已存在则返回 `false`
`public boolean mkdir()`	**创建由此 File 对象表示的目录。**注意：**仅能创建单级目录**，且如果父目录不存在，创建会失败
`public boolean mkdirs()`	**创建由此 File 对象表示的目录，包括任何必需但不存在父目录。**可以用于创建多级目录
`public boolean delete()`	**删除由此 File 对象表示的文件或目录。**注意：**要删除目录，该目录必须为空**
`public String[] list()`	返回一个字符串数组，包含此 File 对象（必须是存在的目录）表示的目录中的**所有文件和子目录的名称**
`public File[] listFiles()`	返回一个 File 对象数组，包含此 File 对象（必须是存在的目录）表示的目录中的**所有文件和子目录的 File 实例**

9.4File类

- 2.遍历目录下的文件

```java
import java.io.*;
public class FileList {
    public static void main(String[] args) {
        File file = new File("D:\\javaprogram\\FJNUObject\\src");
        if(file.isDirectory()) {
            String[] fileNames = file.list();
            for (String name:fileNames)
                System.out.println(name);
        }
    }
}
```

```
CampusManagementSystem.java
ConvertExample.java
EmaiValidator.java
exer1.java
FileIOStreamExample.java
FileList.java
FileRWExample.java
Main.java
MatchObject.java
MathExer.java
StringCst.java
Verify6174.java
```

9.4File类

- 2.遍历目录下的文件(含子目录）

```
import java.io.*;
public class FileList {
    public static void main(String[] args) {
        File file = new File("D:\\javaprogram\\FJNUObject\\src");
        files(file);
    }
    public static void files(File f) {
        File[] files = f.listFiles();
        for(File fi:files) {
            if(fi.isDirectory())
                files(fi);
            System.out.println(fi.getAbsolutePath());
        }
    }
}
```

```
D:\javaprogram\FJNUObject\src\CampusManagementSystem.java
D:\javaprogram\FJNUObject\src\ConvertExample.java
D:\javaprogram\FJNUObject\src\EmaiValidator.java
D:\javaprogram\FJNUObject\src\exer1.java
D:\javaprogram\FJNUObject\src\FileIOStreamExample.java
D:\javaprogram\FJNUObject\src\FileList.java
D:\javaprogram\FJNUObject\src\FileRWExample.java
D:\javaprogram\FJNUObject\src\listFiles\test.txt
D:\javaprogram\FJNUObject\src\listFiles
D:\javaprogram\FJNUObject\src\Main.java
D:\javaprogram\FJNUObject\src\MatchObject.java
D:\javaprogram\FJNUObject\src\MathExer.java
D:\javaprogram\FJNUObject\src\StringCst.java
D:\javaprogram\FJNUObject\src\Verify6174.java
```

9.4File类

- 3.文件过滤

```
import java.io.*;
public class Filter {
    public static void main(String[] args) {
        File file = new File("D:\\javaprogram\\FJNUObject\\src");
        File[] list = file.listFiles(new NameFilter());
        for(File fi:list){
            System.out.println(fi);
        }
    }
}
class NameFilter implements FileFilter{
    public boolean accept(File pathname) {
        String name = pathname.getName();
        return name.endsWith(".java");
    }
    return name.endsWith(".txt");
}
```

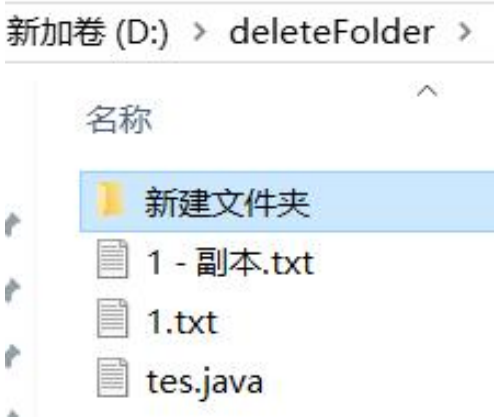
```
D:\javaprogram\FJNUObject\src\CampusManagementSystem.java
D:\javaprogram\FJNUObject\src\ConvertExample.java
D:\javaprogram\FJNUObject\src\EmaiValidator.java
D:\javaprogram\FJNUObject\src\exer1.java
D:\javaprogram\FJNUObject\src\FileIOStreamExample.java
D:\javaprogram\FJNUObject\src\FileList.java
D:\javaprogram\FJNUObject\src\FileRWExample.java
D:\javaprogram\FJNUObject\src\Filter.java
D:\javaprogram\FJNUObject\src\Main.java
D:\javaprogram\FJNUObject\src\MatchObject.java
D:\javaprogram\FJNUObject\src\MathExer.java
D:\javaprogram\FJNUObject\src\StringCst.java
D:\javaprogram\FJNUObject\src\Verify6174.java
```

```
D:\javaprogram\FJNUObject\src\test.txt
```

9.4File类

- 4.删除文件及目录

```
import java.io.*;
public class deleteFile {
    public static void main(String[] args) {
        File file =new File("d:\\deleteFolder");
        deleteFiles(file);
    }
    public static void deleteFiles(File file) {
        if(file.exists()) {
            File[] files = file.listFiles();
            for(File f:files) {
                if(f.isDirectory())
                    deleteFiles(f);
                else {
                    System.out.println("删除了文件"+f.getName());
                    f.delete();
                }
            }
        }
        System.out.println("删除了目录："+file.getName());
        file.delete();
    }
}
```




```
删除了文件1 - 副本.txt
删除了文件1.txt
删除了文件tes.java
删除了目录：新建文件夹
删除了目录：deleteFolder
```

9.5RandomAccessFile类

- 除了File类之外，Java还提供了RandomAccessFile类用于专门处理文件，它支持“随机访问”的方式，这里“随机”是指可以跳转到文件的任意位置处读写数据。使用RandomAccessFile类，程序可以直接跳到文件的任意地方读、写文件，既支持只访问文件的部分内容，又支持向已存在的文件追加内容。