课程目标及安排

- 查询文件夹中的所有子目录
- 文件过滤器
- 递归

文件过滤器的应用

需求

- 获取指定目录下,满足特定条件的文件(例如后缀为xml文件)

解决方案

- 可以使用File类型中的listFiles(FileFilter fileFilter)方法进行实现
- 为了简化代码的编写，可以在代码实现过程中使用lambda表达式

代码实现

- 代码实现

```
package file;

import java.io.File;
import java.io.FileFilter;

public class FileFilterDemo {
    /**
     * 获取项目当前目录下的所有.xml文件
     */
    static void doListFiles01(){
        //1.获取项目的当前目录(这里的"."代表当前目录)
        File dir=new File(".");
        System.out.println(dir.getAbsolutePath());
        //2.假如不是目录则直接返回
        if(!dir.isDirectory())return;
        //3.获取满足条件的文件
        //3.1定义一个文件过滤器?(经常用于按用户需求对文件进行过滤)
        FileFilter fileFilter=new FileFilter() {
            /*此方法的作用就是用于判定文件是否满足我们的要求,满足则应返回true*/
            @Override
            public boolean accept(File file) {
                return
file.isFile()&file.getName().toLowerCase().endsWith(".xml");
            }
        };
        //3.2应用过滤器获取文件
        File[] fileArray=dir.listFiles(fileFilter);
        //3.3 遍历获取的文件
        for(File file:fileArray){
            System.out.println(file.getName());
```

```
        }
    }
    /**
     * 获取项目当前目录下的所有.xml文件
     */
    static void doListFiles02(){
        //1.获取项目的当前目录(这里的"."代表当前目录)
        File dir=new File(".");
        System.out.println(dir.getAbsolutePath());
        //2.假如不是目录则直接返回
        if(!dir.isDirectory())return;
        //3.获取满足条件的文件
        //3.1定义一个文件过滤器?(经常用于按用户需求对文件进行过滤)
        FileFilter fileFilter=new FileFilter() {
            /*此方法的作用就是用于判定文件是否满足我们的要求,满足则应返回true*/
            @Override
            public boolean accept(File file) {
                return
file.isFile()&file.getName().toLowerCase().endsWith(".xml");
            }
        };
        //3.2应用过滤器获取文件(应用lambda表达式)
        File[] fileArray=dir.listFiles(
                file->file.isFile()&
                        file.getName().toLowerCase().endsWith(".xml"));
        //3.3 遍历获取的文件
        for(File file:fileArray){
            System.out.println(file.getName());
        }
    }
    public static void main(String[] args) {
       // doListFiles01();
        doListFiles02();
    }
}
```

递归操作实践

什么是递归

- 递归是一种自身调用自身的逻辑，通常用于处理一些有规律的反复执行的动作。
  注意：递归应该有一个结束条件，假如没有结束条件，可能会出现无限递归，然后导致内存溢出。

递归应用案例分析

- 递归查询指定目录下的文件

```
package file;

import java.io.File;

public class FileScanDemo {
    //无限递归,会出现内存溢出(栈内存溢出-StackOverflowError)
```

```
    static void doScan(){
        doScan();
    }

    /**
     * 获取指定目录以及此目录下所有子目录中的xml文件
     * @param file
     */
    static void doScanXml(File file){
        File[] files=file.listFiles();
        for(File f:files){
            if(f.isDirectory()){
                //自身调用自身
                doScanXml(f);
            }else{
                if(f.getName().endsWith(".xml"))
                System.out.println(f.getAbsolutePath());
            }
        }
    }
    public static void main(String[] args) {
        //doScan();
        doScanXml(new File("."));
    }
}
```

- 递归删除指定目录以及目录下的所有文件和目录

```
package file;

import java.io.File;

/**
 * 思考:删除当前项目目录下的a目录
 */
public class Test3 {
    public static void main(String[] args) {
        File dir = new File("./a");
        delete(dir);
    }
    /**递归删除指定目录下的所有目录和文件*/
    public static void delete(File f){
        if(f.exists()&&f.isDirectory()){
            File[] files = f.listFiles();
            for(File subFile:files){
                if(subFile.isDirectory()){
                    delete(subFile);//递归调用
                }else{
                    subFile.delete();
                }
            }
        }
        f.delete();
    }
```

```
}
```

IO操作实践

何为IO?

- Input(输出): 在程序中用于描述读取数据
- Output(输出)：在程序中用于描述输出数据

何为流?

在编程语言中通过输入输出通常称之为“流”(Stream),类似生活中的水流、电流等。
我们这里要将的流通常指的是二进制数据流，然后通过输入和输出方式进行数据流的读取和写入。

Java中的流超类

- InputStream(字节输入流的超类)
- OutputStream(字节输出流的超类)

Java中的文件流

- FileInputStream (文件字节输入流)
  - read()
  - read(byte[] data)
- FileOutputStream (文件字节输出流)
  - write(int d)
  - write(byte[] data)

文件写操作实践

基于FileOutputStream对象向文件写数据，例如:

```
package io;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;

public class FOSDemo {
    public static void main(String[] args) throws IOException {
        //1.在当前目录构建一个文件输出流对象
        FileOutputStream fos=
                new FileOutputStream("fos.dat");
        //2.向文件中写入三个整数
        fos.write(1);//只写低8位
        fos.write(2);
        fos.write(3);
        //3.关闭流
        fos.close();
```

```
    }
}
```

文件读操作实践

基于FileIntputStream对象向文件写数据，例如:

```
package io;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

public class FISDemo {
    public static void main(String[] args) throws IOException {
        //构建输入流对象
        FileInputStream fis=new FileInputStream("fos.dat");
        //读数据(一次读取一个字节)
        int a = fis.read();
        System.out.println(a);
        int b=fis.read();
        System.out.println(b);
        int c=fis.read();
        System.out.println(c);
        int d=fis.read();//-1表示已经读到文件尾
        System.out.println(d);
        //释放资源
        fis.close();
    }
}
```

文件复制操作实践

基于FileInputStream和FileOutputStream实现文件内容的复制

```
package io;

import com.sun.xml.internal.ws.policy.privateutil.PolicyUtils;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;

/**
 * 文件复制应用案例
 */
public class FileCopyDemo {
    static void doCopy01() throws IOException {
        //1.创建流对象(输入流，输出流)
        FileInputStream fis = new FileInputStream("a.png");
        FileOutputStream fos = new FileOutputStream("b.png");
        //2.读写文件数据(复制)
```

```
        int data;
        while ((data = fis.read()) != -1) {//这里一次读一个字节，当读到文件尾时返回值是-1
            //将读取的数据写入到文件
            fos.write(data);
        }
        //3.释放资源(关闭流)
        fis.close();
        fos.close();
    }

    static void doCopy02() throws IOException {
        //1.创建流对象(输入流，输出流)
        FileInputStream fis = new FileInputStream("a.png");
        FileOutputStream fos = new FileOutputStream("b.png");
        //2.读写文件数据(复制)
        byte[] buf = new byte[1024];//通过此数据存储一次读取的字节数
        int len=0;
        while ((len = fis.read(buf)) != -1) {//这里一次读取多个字节,len为读取的字节数
            //将读取的字节写入到文件,0表示起始位置，len表示写长度
            fos.write(buf, 0, len);
            //假如有页面，在这个位置要返回刷新页面进度条的数据
        }
        //3.释放资源(关闭流)
        fis.close();
        fos.close();
    }
    public static void main(String[] args) throws IOException {
        long t1 = System.currentTimeMillis();
        doCopy02();
        long t2 = System.currentTimeMillis();
        //输出耗时时长
        System.out.println(t2 - t1);
    }
}
```

文本内容操作实践

基于FileInputStream和FileOutputStream对字符串内容进行读写

```
package io;

import java.io.*;
import java.nio.charset.StandardCharsets;

public class FileTextDemo {
    static void doWrite()throws IOException{
        String s1="我爱Java";
        FileOutputStream fos=new FileOutputStream("f1.txt",true);//true表示追加
        fos.write(s1.getBytes(StandardCharsets.UTF_8));
        fos.close();
        System.out.println("write ok");
    }
```

```
    static void doRead01()throws IOException{
        //1.构建文件对象
        File file=new File("f1.txt");
        //2.构建字节输入流对象
        FileInputStream fis = new FileInputStream(file);
        //3.读取数据
        //3.1构建字节数组，用于存储读取的字节
        byte[] data=new byte[(int)file.length()];
        //3.2将流中的数据读取到字节数组中
        fis.read(data);
        //4.释放资源
        fis.close();
        //5.构建字符文本对象
        String s = new String(data, StandardCharsets.UTF_8);
        System.out.println(s);
    }
    static void doRead02()throws IOException{
        //1.构建字节输入流对象
        FileInputStream fis = new FileInputStream("f1.txt");
        //2.读取数据
        //2.1构建字节数组，用于存储读取的字节
        byte[] data=new byte[fis.available()];//available方法用于返回流中可读取的字节
数
        //2.2将流中的数据读取到字节数组中
        fis.read(data);
        //3.释放资源
        fis.close();
        //4.构建字符文本对象
        String s = new String(data, StandardCharsets.UTF_8);
        System.out.println(s);
    }
    public static void main(String[] args) throws IOException {
        doRead02();
    }
}
```

总结(summary)

File类

File类的每一个实例可以表示硬盘(文件系统)中的一个文件或目录(实际上表示的是一个抽象路径)

使用File可以做到:

1:访问其表示的文件或目录的属性信息,例如:名字,大小,修改时间等等
2:创建和删除文件或目录
3:访问一个目录中的子项

常用构造器:

```
File(String pathname)
File(File parent,String name)可参考文档了解
```

常用方法:

length()：返回一个long值，表示占用的磁盘空间，单位为字节。
canRead()：File表示的文件或目录是否可读
canWrite()：File表示的文件或目录是否可写
isHidden()：File表示的文件或目录是否为隐藏的
createNewFile()：创建一个新文件，如果指定的文件所在的目录不存在会抛出异常
java.io.FileNotFoundException
mkdir：创建一个目录
mkdirs：创建一个目录，并且会将所有不存在的父目录一同创建出来，推荐使用。
delete()：删除当前文件或目录，如果目录不是空的则删除失败。
exists()：判断File表示的文件或目录是否真实存在。true:存在 false:不存在
isFile()：判断当前File表示的是否为一个文件。
isDirectory()：判断当前File表示的是否为一个目录
listFiles()：获取File表示的目录中的所有子项
listFiles(FileFilter filter)：获取File表示的目录中满足filter过滤器要求的所有子项

JAVA IO必会概念

java io可以让我们用标准的读写操作来完成对不同设备的读写数据工作.
java将IO按照方向划分为输入与输出,参照点是我们写的程序.
输入:用来读取数据的,是从外界到程序的方向,用于获取数据.
输出:用来写出数据的,是从程序到外界的方向,用于发送数据.
java将IO比喻为"流",即:stream. 就像生活中的"电流","水流"一样,它是以同一个方向顺序移动的过程.只不过这里流动的是字节(2进制数据).所以在IO中有输入流和输出流之分,我们理解他们是连接程序与另一端的"管道",用于获取或发送数据到另一端.

因此流的读写是顺序读写的，只能顺序向后写或向后读，不能回退。

Java两个IO超类(抽象类)

java.io.InputStream:所有字节输入流的超类,其中定义了读取数据的方法.因此将来不管读取的是什么设备(连接该设备的流)都有这些读取的方法,因此我们可以用相同的方法读取不同设备中的数据

常用方法:

int read()：读取一个字节，返回的int值低8位为读取的数据。如果返回值为整数-1则表示读取到了流的末尾

int read(byte[] data)：块读取，最多读取data数组总长度的数据并从数组第一个位置开始存入到数组中，返回值表示实际读取到的字节量，如果返回值为-1表示本次没有读取到任何数据，是流的末尾。
java.io.OutputStream:所有字节输出流的超类,其中定义了写出数据的方法.

常用方法:

```
void write(int d)：写出一个字节，写出的是给定的int值对应2进制的低八位。

void write(byte[] data)：块写，将给定字节数组中所有字节一次性写出。

void write(byte[]data,int off,int len)：块写，将给定字节数组从下标off处开始的连续len个字节一次性写出。
```

文件输出流

java.io.FileOutputStream文件输出流，继承自java.io.OutputStream
常用构造器
覆盖模式对应的构造器
覆盖模式是指若指定的文件存在，文件流在创建时会先将该文件原内容清除。
FileOutputStream(String pathname)：创建文件输出流用于向指定路径表示的文件做写操作

FileOutputStream(File file)：创建文件输出流用于向File表示的文件做写操作。

注:如果写出的文件不存在文件流自动创建这个文件，但是如果该文件所在的目录不存在会抛出异常:java.io.FileNotFoundException

追加写模式对应的构造器
追加模式是指若指定的文件存在，文件流会将写出的数据陆续追加到文件中。
FileOutputStream(String pathname,boolean append)：如果第二个参数为true则为追加模式，false则为覆盖模式
FileOutputStream(File file,boolean append)：同上
常用方法:
void write(int d)：向文件中写入一个字节，写入的是int值2进制的低八位。

void write(byte[] data)：向文件中块写数据。将数组data中所有字节一次性写入文件。

void write(byte[] data,int off,int len)：向文件中快写数据。将数组data中从下标off开始的连续len个字节一次性写入文件。

文件输入流

java.io.FileInputStream文件输入流，继承自java.io.InputStream
常用构造器
FileInputStream(String pathname) 创建读取指定路径下对应的文件的文件输入流，如果指定的文件不存在则会抛出异常java.io.FileNotFoundException

FileInputStream(File file) 创建读取File表示的文件的文件输入流，如果File表示的文件不存在则会抛出异常java.io.IOException。

常用方法
int read()：从文件中读取一个字节，返回的int值低八位有效，如果返回的int值为整数-1则表示读取到了文件末尾。

int read(byte[] data)：块读数据，从文件中一次性读取给定的data数组总长度的字节量并从数组第一个元素位置开始存入数组中。返回值为实际读取到的字节数。如果返回值为整数-1则表示读取到了文件末尾。