📖 青空的b站课程

📅 2024年4月20日

文件字节流

输入流

⭐ `FileInputStream`,可以使用它来获取文件的输入流

```
public static void main(String[] args) {
    try {   //注意，IO相关操作会有很多影响因素，有可能出现异常，所以需要明确进行处理
        FileInputStream inputStream = new FileInputStream("路径");
        //路径支持相对路径和绝对路径
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}
```

1️⃣ 当你读取完成一个文件的时候，就需要使用`close`来关闭流，否则会一直占用那个文件，导致其他程序在读取这个文件的时候就会显示占用，但是如果直接写在try语句内，就会发现，假如中间的代码发生异常救会报错，从而不执行close，导致资源一直被占用

```
    public static void main(String[] args) {
        try {
            FileInputStream inputStream = new FileInputStream("路径");
            //... 假如中间的代码发生异常救会报错，从而不执行close
            inputStream.close();
        } catch (IOException e) { //这里可以直接使用Io异常，文件异常就是继承于这个
            throw new RuntimeException(e);
        }
    }
```

2️⃣ 可以使用`finally`方法，但是这样显得有点繁琐了

```
    public static void main(String[] args) {
        FileInputStream inputStream = null;
        try {
            inputStream = new
FileInputStream("D:\\learnCode\\Java\\JavaCollection\\JavaCollection\\src\\main\
\java\\fun\\tanc\\test.txt");
            System.out.println(inputStream.read());

        } catch (IOException e) {
            throw new RuntimeException(e);
        }finally {
            try {
                inputStream.close(); //close可能也会抛异常，所以也需要try catch
            } catch (IOException e) {
```

```
                throw new RuntimeException(e);
            }
        }
    }
```

3️⃣ 有个更加方便便捷的语句，也就是一个语法糖，它可以将上面写的代码简化

```
public static void main(String[] args) {

    //注意，这种语法只支持实现了AutoCloseable接口的类!
    try(FileInputStream inputStream = new FileInputStream("路径")) {   //直接在try()中定义要在完成之后释放的资源

    } catch (IOException e) {   //这里变成IOException是因为调用close()可能会出现，而FileNotFoundException是继承自IOException的
        e.printStackTrace();
    }
    //无需再编写finally语句块，因为在最后自动帮我们调用了close()
}
```

⭐ 使用`read`来读取文件，注意`read`读取出来的是字节值并不是它的原始数据，需要使用`char`强转

```
   public static void main(String[] args) {
       try(FileInputStream inputStream = new FileInputStream("D:\\learnCode\\Java\\JavaCollection\\JavaCollection\\src\\main\\java\\fun\\tanc\\test.txt");) {
           System.out.println((char) inputStream.read());
       } catch (IOException e) {
           throw new RuntimeException(e);
       }
   }
```

1️⃣ 它返回`-1`表示文件以及读取完成了

```
public static void main(String[] args) {
    //test.txt: a
    try(FileInputStream inputStream = new FileInputStream("test.txt")) {
        //使用read()方法进行字符读取
        System.out.println((char) inputStream.read());  //读取一个字节的数据（英文字母只占1字节，中文占2字节）
        System.out.println(inputStream.read());   //唯一一个字节的内容已经读完了，再次读取返回-1表示没有内容了
    }catch (IOException e){
        e.printStackTrace();
    }
}
```

⭐ `read`方法只能一下读取一个字符，那如果我们想要将文件读取完成就需要使用`while`语句

```
    public static void main(String[] args) {
        try(FileInputStream inputStream = new FileInputStream("D:\\learnCode\\Java\\JavaCollection\\JavaCollection\\src\\main\\java\\fun\\tanc\\test.txt");) {
            int i;
            while ((i = inputStream.read()) != -1){
                System.out.print((char) i);
            }
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
```

⭐ 使用`available`方法能查看当前可读的剩余字节数量（注意：并不一定真实的数据量就是这么多，尤其是在网络I/O操作时，这个方法只能进行一个预估也可以说是暂时能一次性可以读取的数量，当然在磁盘IO下，一般情况都是真实的数据量）

```
    public static void main(String[] args) {
        try(FileInputStream inputStream = new FileInputStream("D:\\learnCode\\Java\\JavaCollection\\JavaCollection\\src\\main\\java\\fun\\tanc\\test.txt");) {
            System.out.println(inputStream.available()); //查看剩余读取的字符数量
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
```

⭐ 上面那种方法是一个一个读取的，读取效率太低了，可以使用`byte[]`数组来直接读取一个数组长度的字符，可以配合`available`来使用，使用read读取的内容会存放到`byte`数组中，存放的也是它的字节值，最后通过`String`内部的构造方法传入一个`byte`数组来直接输出；

⚠ 一次性读取同单个读取一样，当没有任何数据可读时，依然会返回-1

```
public static void main(String[] args) {
    //test.txt：abcd
    try(FileInputStream inputStream = new FileInputStream("test.txt")) {
        byte[] bytes = new byte[inputStream.available()];   //我们可以提前准备好合适容量的byte数组来存放
        System.out.println(inputStream.read(bytes));   //一次性读取全部内容（返回值是读取的字节数）
        System.out.println(new String(bytes));   //通过String(byte[])构造方法得到字符串
    }catch (IOException e){
        e.printStackTrace();
    }
}
```

1️⃣ 这样还可以限制只读取哪几个字节

```
System.out.println(inputStream.read(bytes,1,3));
```

⭐ 可以通过`skip`来跳过字符

```
public static void main(String[] args) {
    //test.txt: abcd
    try(FileInputStream inputStream = new FileInputStream("test.txt")) {
        System.out.println(inputStream.skip(1));
        System.out.println((char) inputStream.read());    //跳过了一个字节
    }catch (IOException e){
        e.printStackTrace();
    }
}
```

输出流

⭐ 和输入流差不多，输出流使用`FileOutputStream`，使用方法也和输入流类似，只不过，它是`write`不是`read`

1️⃣ `write`也只能一次写一个字符，但是写中文的时候会有些问题还是

```
    public static void main(String[] args) {
        try(FileOutputStream outputStream = new FileOutputStream("test.txt")){
            outputStream.write('h');
        }catch (IOException e){
            e.printStackTrace();
        }
    }
```

2️⃣ 它内部也可以添加`byte`数组来达到输入一个字符串的效果，这里演示的是从一个文件的内容拷贝到另外一个文件的内容

```
    public static void main(String[] args) {
        /*
        * test.txt没有东西
        *
D:\learnCode\Java\JavaCollection\JavaCollection\src\main\java\fun\tanc\test.txt内存储的hello World!
        * */
        //定义了两个流一个输入一个输出
        try(FileOutputStream outputStream = new FileOutputStream("test.txt");
            FileInputStream inputStream = new
FileInputStream("D:\\learnCode\\Java\\JavaCollection\\JavaCollection\\src\\main\\java\\fun\\tanc\\test.txt"))
        {
            byte[] b = new byte[12]; //创建数组
            inputStream.read(b);  //使用输出流来将文件数据读取到数组
            outputStream.write(b); //写入数组内的数据到文件
```

```
        }catch (IOException e){
            e.printStackTrace();
        }
    }
```

3️⃣ 当然还有个更加方便的办法使用`getBytes`直接转换为`Byte`数组

```
    public static void main(String[] args) {
        /*
        * test.txt没有东西
        *
D:\learnCode\Java\JavaCollection\JavaCollection\src\main\java\fun\tanc\test.txt
内存储的hello World!
        * */
        try(FileOutputStream outputStream = new
FileOutputStream("test.txt",true)) {
            outputStream.write("HelloWorld!".getBytes());
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
```

⭐ 但是此时你的写入是覆盖的，并不会追加，如果需要追加需要使用`FileOutputStream`的另外一个构造方法

```
//源代码
public FileOutputStream(String name, boolean append) //apped表示是否追加默认是false
        throws FileNotFoundException
    {
        this(name != null ? new File(name) : null, append);
    }
```

1️⃣ 代码实列

```
    public static void main(String[] args) {
        /*
        * test.txt没有东西
        *
D:\learnCode\Java\JavaCollection\JavaCollection\src\main\java\fun\tanc\test.txt
内存储的hello World!
        * */
        try(FileOutputStream outputStream = new
FileOutputStream("test.txt",true)) { //true表示追加
            outputStream.write("HelloWorld!".getBytes());
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
```

⭐ 同样的`write`也可以限制写入的字符长度

```
outputStream.write(b, 0, 1)
```

整体方法查询

当然，以下是一个关于Java中`FileInputStream`和`FileOutputStream`类（文件字节流）的方法及使用说明的表格：

类名	方法名称	描述
`FileInputStream`		
构造方法	`FileInputStream(File file)`	创建一个`FileInputStream`对象，用于从指定File对象表示的文件读取字节
	`FileInputStream(String path)`	创建一个`FileInputStream`对象，用于从指定路径的文件读取字节
读取方法	`int read()`	读取一个字节，返回值为读取到的字节（0-255）或-1（到达文件末尾）
	`int read(byte[] b)`	读取若干字节到字节数组b中，返回实际读取的字节数
	`int read(byte[] b, int off, int len)`	从文件读取`len`个字节到b数组的off偏移量处开始的位置
关闭方法	`void close()`	关闭此文件输入流并释放与之关联的所有系统资源

类名	方法名称	描述
`FileOutputStream`		
构造方法	`FileOutputStream(File file)`	创建一个`FileOutputStream`对象，用于向指定File对象表示的文件写入字节
	`FileOutputStream(String path)`	创建一个`FileOutputStream`对象，用于向指定路径的文件写入字节
写入方法	`void write(int b)`	将指定的字节写入此输出流
	`void write(byte[] b)`	将指定字节数组中的所有字节写入此输出流
	`void write(byte[] b, int off, int len)`	将指定字节数组的一部分（从off偏移量开始，长度为`len`）写入此输出流
关闭方法	`void close()`	关闭此文件输出流并释放与此流相关联的所有系统资源

此外，还有结合这两个类实现文件复制的典型示例：

```
public void copyFileUsingStream(String source, String destination) throws
IOException {
    FileInputStream in = new FileInputStream(source);
    FileOutputStream out = new FileOutputStream(destination);

    byte[] buffer = new byte[1024];
    int length;

    while ((length = in.read(buffer)) > 0) {
        out.write(buffer, 0, length);
    }

    in.close();
    out.close();
}
```

在这个例子中，首先创建`FileInputStream`和`FileOutputStream`对象分别指向源文件和目标文件，然后读取字节流并写入到另一个字节流中，直至文件结束。最后，关闭两个流以释放资源。