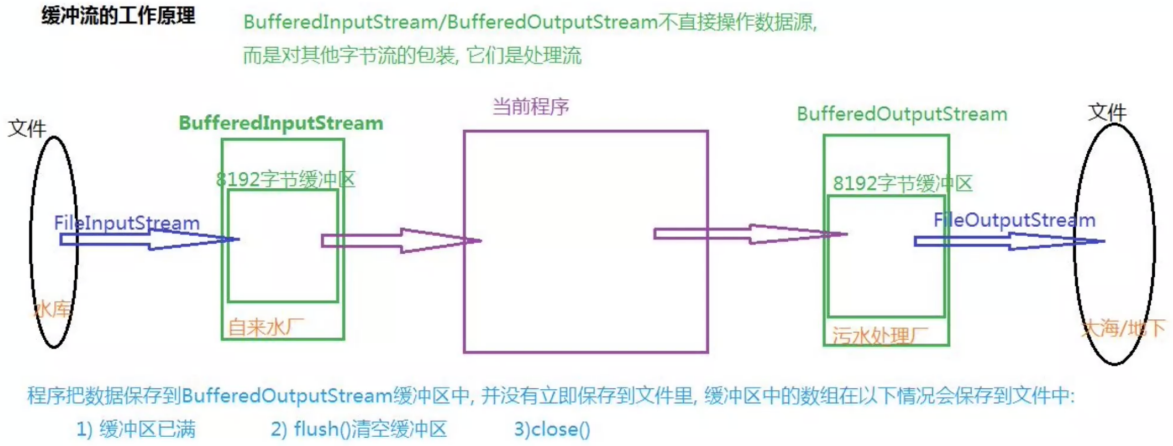
📖 青空的b站课程

📅 2024年4月22日

缓存流

⭐ 虽然普通的文件流读取文件数据非常便捷，但是每次都需要从外部I/O设备去获取数据，由于外部I/O设备的速度一般都达不到内存的读取速度，很有可能造成程序反应迟钝，因此性能还不够高，而缓冲流正如其名称一样，它能够提供一个缓冲，提前将部分内容存入内存（缓冲区）在下次读取时，如果缓冲区中存在此数据，则无需再去请求外部设备。同理，当向外部设备写入数据时，也是由缓冲区处理，而不是直接向外部设备写入。



缓冲输入

⭐ 如果要创建一个缓存字节流，需要将之前输入/输出流写入缓存流即可，操作和之前的也差不多

```
class Main {
    public static void main(String[] args) {
        //将普通流加一个缓冲流
        try(BufferedInputStream bufferedInputStream = new BufferedInputStream(
new FileInputStream("test.txt"))){
             System.out.println((char) bufferedInputStream.read());
        }catch (IOException e){
            throw  new RuntimeException();
        }
    }
}
```

1️⃣ 查看一下缓存输入流的源代码，它内部提供了一个缓存去大小，暂存数据

```
public
class BufferedInputStream extends FilterInputStream {

    private static int DEFAULT_BUFFER_SIZE = 8192; //默认缓存大小8192

    private static int MAX_BUFFER_SIZE = Integer.MAX_VALUE - 8; //最大缓存值

    protected volatile byte buf[]; //这是一个专门用于缓存的数组
    .....
}
```

2️⃣ 仔细观察就会发现，有点像`HashSet`一样替身攻击，它也是直接在代码内部创建了一个`InputStream`对象，然后使用这个对象来做一些操作，实际上进行`IO`操作的还是传入的`FileInputStream`，这种替身模式叫做`装饰者模式`

```
    public void close() throws IOException {
        byte[] buffer;
        while ( (buffer = buf) != null) {
            if (bufUpdater.compareAndSet(this, buffer, null)) {
                InputStream input = in;
                in = null;
                if (input != null)
                    input.close();
                return;
            }
            // Else retry in case a new buf was CASed in fill()
        }
    }
```

⭐ I/O操作一般不能重复读取内容（比如键盘发送的信号，主机接收了就没了），而缓冲流提供了缓冲机制，一部分内容可以被暂时保存，`BufferedInputStream`支持`reset()`和`mark()`操作，首先我们来看看`mark()`方法的介绍

```
/**
*标记输入流中的当前位置。随后
*调用<code>reset</code>方法会将该流重新定位到
*最后标记的位置，以便后续读操作重新读取相同的字节。
* < p >
* <code>readlimit</code>参数告诉输入流
*允许在获取标记位置之前读取这么多字节
*失效。
 * <p>
 * This method simply performs <code>in.mark(readlimit)</code>.
 *
 * @param   readlimit   the maximum limit of bytes that can be read before
 *                      the mark position becomes invalid.
 * @see     java.io.FilterInputStream#in
 * @see     java.io.FilterInputStream#reset()
 */
public synchronized void mark(int readlimit) {
```

```
    in.mark(readlimit);
}
```

1 实际上是对读取的位置打一个`mark`,它需要接受一个`readlimt`，入流会以某种方式保留之后读取的`readlimit`数量的内容，当读取的内容数量超过`readlimit`则之后的内容不会被保留，当调用`reset()`之后，会使得当前的读取位置回到`mark()`调用时的位置。

```
    public static void main(String[] args) {
        //文件内字符为HelloWorld!
        //将普通流加一个缓冲流
        try(BufferedInputStream bufferedInputStream = new BufferedInputStream(new FileInputStream("test.txt"))){
            bufferedInputStream.mark(3);
            System.out.print((char) bufferedInputStream.read()); //H
            System.out.print((char) bufferedInputStream.read()); //e
            System.out.print((char) bufferedInputStream.read()); //l
            System.out.println((char) bufferedInputStream.read()); //l
            bufferedInputStream.reset();
            System.out.print((char) bufferedInputStream.read()); //到这里就继续输出H并没有继续输出o
            System.out.print((char) bufferedInputStream.read()); //从头到尾继续输出
            System.out.print((char) bufferedInputStream.read());
            System.out.print((char) bufferedInputStream.read());
            System.out.print((char) bufferedInputStream.read());

        }catch (IOException e){
            throw  new RuntimeException();
        }
    }
```

2 以上的实例并没有实现`readLimt`限制，还可以继续输出，这是因为默认情况下缓存为`8192`，上面填个`3`肯定小于，它的构造方法可以添加一个替换默认缓存大小

```
   public BufferedInputStream(InputStream in, int size) {
        super(in);
        if (size <= 0) {
            throw new IllegalArgumentException("Buffer size <= 0");
        }
        buf = new byte[size];
    }
```

我们加上后在再去执行，会报错，表示这个`mark`失效

```
    public static void main(String[] args) {
        try(BufferedInputStream bufferedInputStream = new BufferedInputStream(new FileInputStream("test.txt"),3)){ //将缓存大小限制为3
            bufferedInputStream.mark(3);
            System.out.print((char) bufferedInputStream.read());
            System.out.print((char) bufferedInputStream.read());
```

```
            System.out.print((char) bufferedInputStream.read());
            System.out.println((char) bufferedInputStream.read());
            bufferedInputStream.reset(); //由于read读取已经操作了3，mark标记失效报错
            System.out.print((char) bufferedInputStream.read());
            System.out.print((char) bufferedInputStream.read());
            System.out.print((char) bufferedInputStream.read());
            System.out.print((char) bufferedInputStream.read());
            System.out.print((char) bufferedInputStream.read());

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

```
Hell
java.io.IOException Create breakpoint : Resetting to invalid mark
    at java.io.BufferedInputStream.reset(BufferedInputStream.java:448)
    at fun.tanc.Main.main(Main.java:18)
```

这样就可以

```
    public static void main(String[] args) {
        //将普通流加一个缓冲流
        try(BufferedInputStream bufferedInputStream = new BufferedInputStream(
new FileInputStream("test.txt"),3)){
            bufferedInputStream.mark(3);
            System.out.print((char) bufferedInputStream.read());
            System.out.print((char) bufferedInputStream.read());
            System.out.print((char) bufferedInputStream.read());
//            System.out.println((char) bufferedInputStream.read());
            bufferedInputStream.reset();
            System.out.print((char) bufferedInputStream.read());
            System.out.print((char) bufferedInputStream.read());
            System.out.print((char) bufferedInputStream.read());
            System.out.print((char) bufferedInputStream.read());
            System.out.print((char) bufferedInputStream.read());

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

缓冲输出

⭐ 缓冲输出流和输入流使用方法大差不差，不过是替换成`FileOutputStream`了，实现也和缓冲输入流差不多，也是在输出的时候加了一个缓冲区，存储数据

```
    public BufferedOutputStream(OutputStream out, int size) {
        super(out); //也是装饰者模式
        if (size <= 0) {
            throw new IllegalArgumentException("Buffer size <= 0");
        }
        buf = new byte[size];
    }
```

⭐ 使用也差不多

```
    public static void main(String[] args) {
        //将普通流加一个缓冲流
        try(BufferedOutputStream bufferedOutputStream = new BufferedOutputStream(
new FileOutputStream("test.txt"),3)){
            bufferedOutputStream.write('h');
            bufferedOutputStream.flush();
        }catch (IOException e){
            e.printStackTrace();
        }
    }
```

缓冲字符流

⭐ 缓冲字符流和缓冲字符流的逻辑差不多，都是在外面在加一个缓冲区，不同的是字符流需要添加的是`Reader`对象

```
    public static void main(String[] args) {
        //将普通流加一个缓冲流
        try(BufferedReader bufferedReader = new BufferedReader(new
FileReader("test.txt"))){
            System.out.println((char) bufferedReader.read());
        }catch (IOException e){
            e.printStackTrace();
        }
    }
```

⭐ 但是她比字节流方便一点点，它可以按照行来读取，读取的是一串字符串

```
    public static void main(String[] args) {
        //将普通流加一个缓冲流
        try(BufferedReader bufferedReader = new BufferedReader(new
FileReader("test.txt"))){
            System.out.println( bufferedReader.readLine());
            System.out.println( bufferedReader.readLine());
            System.out.println( bufferedReader.readLine());
        }catch (IOException e){
            e.printStackTrace();
        }
    }
```

⭐ 还可以对读取行然后转换为流，来操作，使用`filter`等等

```
    public static void main(String[] args) {
        try (BufferedReader reader = new BufferedReader(new
FileReader("test.txt"))){
            reader
                    .lines()
                    .filter(str -> str.charAt(0) >= 'a' && str.charAt(0)<='z')
                    .limit(2)
                    .distinct()
                    .sorted()
                    .forEach(System.out::println);
        }catch (IOException e) {
            e.printStackTrace();
        }
    }
```

⭐ 它也同样支持`mark`和`rest`操作

```
public static void main(String[] args) {
    try (BufferedReader reader = new BufferedReader(new FileReader("test.txt")))
{
        reader.mark(1);
        System.out.println((char) reader.read());
        reader.reset();
        System.out.println((char) reader.read());
    }catch (IOException e) {
        e.printStackTrace();
    }
}
```

⭐ `BufferedReader`处理纯文本文件时就更加方便了，`BufferedWriter`在处理时也同样方便：

```
public static void main(String[] args) {
    try (BufferedWriter reader = new BufferedWriter(new
FileWriter("output.txt"))){
        reader.newLine();   //使用newLine进行换行
        reader.write("汉堡做滴彳亍不彳亍");   //可以直接写入一个字符串
        reader.flush();   //清空缓冲区
    }catch (IOException e) {
        e.printStackTrace();
    }
}
```