赵彦军

做一只快乐的程序猿！

个人资料

赵彦军

原创	粉丝	喜欢	评论
265	94	42	27

等级： 博客 5　　访问量： 15万+

积分： 2259　　排名： 2万+

文章搜索

博客专栏

CentOS 初体验
文章：25篇
阅读：4976

Gradle 实战指南
文章：7篇
阅读：8792

Fiddler 实战指南
文章：11篇
阅读：15144

redis 实战指南
文章：7篇
阅读：595

文章分类

android (60)
android studio (42)
网站与服务器 (5)
运营与测试 (1)
Java (12)
设计模式 (1)
CVS(Git 、SVN) (1)
工具 (12)
centos (26)
Fiddler (11)

文章存档

2018年3月 (1)
2018年2月 (7)
2018年1月 (15)
2017年12月 (3)
2017年11月 (2)

Java IO流学习总结三：缓冲流-BufferedInputStream、BufferedOutputStream

标签： zhaoyanjun java io流 赵彦军

2017年02月06日 16:41:04　　2316人阅读　　评论(0)　　收藏　　举报

分类： android (59)　Java (11)

Java IO流学习总结三：缓冲流-BufferedInputStream、BufferedOutputStream

> 转载请标明出处：http://blog.csdn.net/zhaoyanjun6/article/details/54894451
> 本文出自【赵彦军的博客】

```
InputStream
|__FilterInputStream
        |__BufferedInputStream
```

首先抛出一个问题，有了`InputStream`为什么还要有`BufferedInputStream`？

`BufferedInputStream`和`BufferedOutputStream`这两个类分别是`FilterInputStream`和`FilterOutputStream`的子类，作为装饰器子类，使用它们可以防止每次读取/发送数据时进行实际的写操作，代表着使用缓冲区。

我们有必要知道不带缓冲的操作，每读一个字节就要写入一个字节，由于涉及磁盘的IO操作相比内存的操作要慢很多，所以不带缓冲的流效率很低。带缓冲的流，可以一次读很多字节，但不向磁盘中写入，只是先放到内存里。等凑够了缓冲区大小的时候一次性写入磁盘，这种方式可以减少磁盘操作次数，速度就会提高很多！

同时正因为它们实现了缓冲功能，所以要注意在使用`BufferedOutputStream`写完数据后，要调用`flush()`方法或`close()`方法，强行将缓冲区中的数据写出。否则可能无法写出数据。与之相似还`BufferedReader`和`BufferedWriter`两个类。

现在就可以回答在本文的开头提出的问题：

`BufferedInputStream`和`BufferedOutputStream`类就是实现了缓冲功能的输入流/输出流。使用带缓冲的输入输出流，效率更高，速度更快。

总结：

- `BufferedInputStream` 是缓冲输入流。它继承于`FilterInputStream`。
- `BufferedInputStream` 的作用是为另一个输入流添加一些功能，例如，提供“缓冲功能”以及支持`mark()`标记和`reset()`重置方法。
- `BufferedInputStream` 本质上是通过一个内部缓冲区数组实现的。例如，在新建某输入流对应的`BufferedInputStream`后，当我们通过`read()`读取输入流的数据时，`BufferedInputStream`会将该输入流的数据分批的填入到缓冲区中。每当缓冲区中的数据被读完之后，输入流会再次填充数据缓冲区；如此反复，直到我们读完输入流数据位置。

BufferedInputStream API简介

源码关键字段分析

```
private static int defaultBufferSize = 8192;//内置缓存字节数组的大小 8KB

protected volatile byte buf[];  //内置缓存字节数组

protected int count;    //当前buf中的字节总数、注意不是底层字节输入流的源中字节总数

protected int pos;      //当前buf中下一个被读取的字节下标

protected int markpos = -1; //最后一次调用mark(int readLimit)方法记录的buf中下一个被读取的字节的位
```

```
11
12 protected int marklimit;    //调用mark后、在后续调用reset()方法失败之前云寻的从in中读取的最大数据量、
13
```

构造函数

```
1 BufferedInputStream(InputStream in) //使用默认buf大小、底层字节输入流构建bis
2
3 BufferedInputStream(InputStream in, int size) //使用指定buf大小、底层字节输入流构建bis
```

一般方法介绍

```
1  int available();  //返回底层流对应的源中有效可供读取的字节数
2
3  void close();  //关闭此流、释放与此流有关的所有资源
4
5  boolean markSupport();  //查看此流是否支持mark
6
7  void mark(int readLimit); //标记当前buf中读取下一个字节的下标
8
9  int read();  //读取buf中下一个字节
10
11 int read(byte[] b, int off, int len);  //读取buf中下一个字节
12
13 void reset();   //重置最后一次调用mark标记的buf中的位子
14
15 long skip(long n);  //跳过n个字节、 不仅仅是buf中的有效字节、也包括in的源中的字节
```

BufferedOutputStream API简介

关键字段

```
1 protected byte[] buf;   //内置缓存字节数组、用于存放程序要写入out的字节
2
3 protected int count;   //内置缓存字节数组中现有字节总数
```

构造函数

```
1 BufferedOutputStream(OutputStream out); //使用默认大小、底层字节输出流构造bos。默认缓冲大小是 8192
2
3 BufferedOutputStream(OutputStream out, int size);  //使用指定大小、底层字节输出流构造bos
```

构造函数源码：

```
1  /**
2    * Creates a new buffered output stream to write data to the
3    * specified underlying output stream.
4    * @param   out   the underlying output stream.
5    */
6   public BufferedOutputStream(OutputStream out) {
7       this(out, 8192);
8   }
9
10  /**
11   * Creates a new buffered output stream to write data to the
12   * specified underlying output stream with the specified buffer
13   * size.
14   *
15   * @param   out    the underlying output stream.
16   * @param   size   the buffer size.
17   * @exception IllegalArgumentException if size &lt;= 0.
18   */
19  public BufferedOutputStream(OutputStream out, int size) {
20      super(out);
21      if (size <= 0) {
22          throw new IllegalArgumentException("Buffer size <= 0");
23      }
24      buf = new byte[size];
25  }
```

一般方法

```
1 //在这里提一句,`BufferedOutputStream`没有自己的`close`方法,当他调用父类`FilterOutputStrem`的方法关闭
2
3 void  flush();  将写入bos中的数据flush到out指定的目的地中、注意这里不是flush到out中、因为其内部又调用
4
5 write(byte b);      将一个字节写入到buf中
6
```

```
write(byte[] b, int off, int len);      将b的一部分写入buf中
```

那么什么时候flush()才有效呢？
答案是：当OutputStream是BufferedOutputStream时。

当写文件需要flush()的效果时，需要
FileOutputStream fos = new FileOutputStream(“c:\a.txt”);
BufferedOutputStream bos = new BufferedOutputStream(fos);
也就是说，需要将FileOutputStream作为BufferedOutputStream构造函数的参数传入，然后对BufferedOutputStream进行写入操作，才能利用缓冲及flush()。

查看BufferedOutputStream的源代码，发现所谓的buffer其实就是一个byte[]。
BufferedOutputStream的每一次write其实是将内容写入byte[]，当buffer容量到达上限时，会触发真正的磁盘写入。
而另一种触发磁盘写入的办法就是调用flush()了。

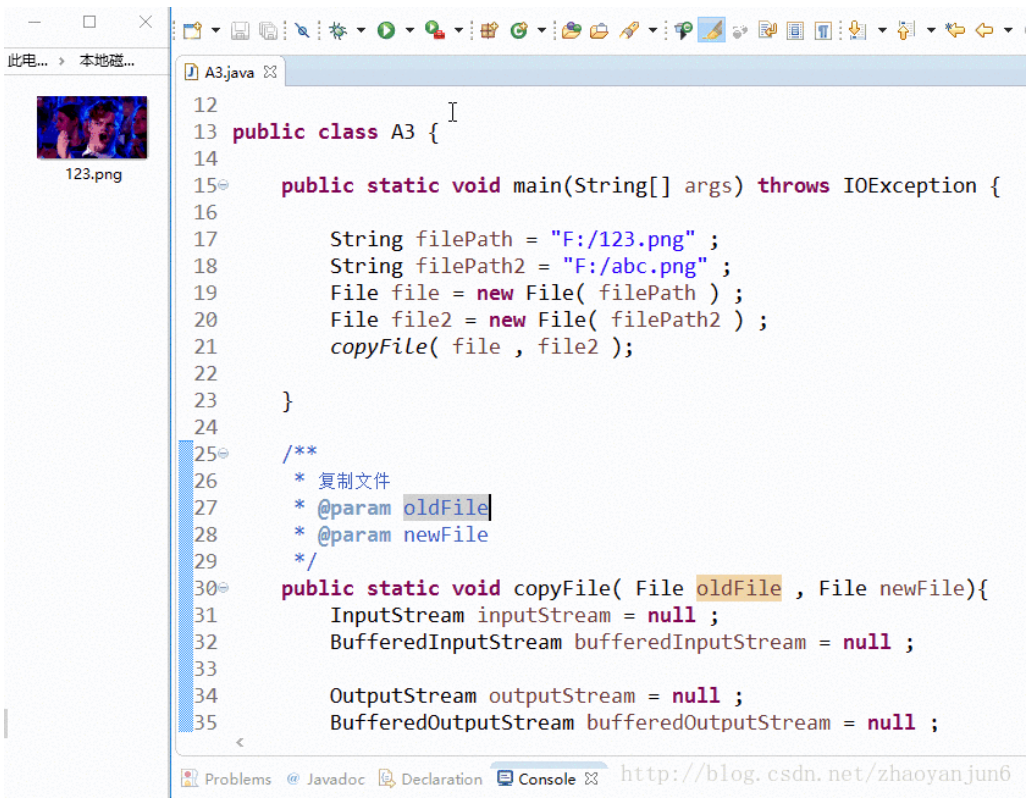
1. `BufferedOutputStream`在`close()`时会自动flush
2. `BufferedOutputStream`在不调用`close()`的情况下，缓冲区不满，又需要把缓冲区的内容写入到文件或通过网络发送到别的机器时，才需要调用flush.

实战演练1：复制文件.
操作：使用缓存流将F盘根目录里面名字为：123.png 图片复制成 abc.png

```
package com.app;
import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;


public class A3 {

    public static void main(String[] args) throws IOException {

        String filePath = "F:/123.png" ;
        String filePath2 = "F:/abc.png" ;
        File file = new File( filePath ) ;
        File file2 = new File( filePath2 ) ;
        copyFile( file , file2 );

    }

    /**
     * 复制文件
     * @param oldFile
     * @param newFile
     */
    public static void copyFile( File oldFile , File newFile){
        InputStream inputStream = null ;
        BufferedInputStream bufferedInputStream = null ;

        OutputStream outputStream = null ;
        BufferedOutputStream bufferedOutputStream = null ;

        try {
            inputStream = new FileInputStream( oldFile ) ;
            bufferedInputStream = new BufferedInputStream( inputStream ) ;

            outputStream = new FileOutputStream( newFile ) ;
            bufferedOutputStream = new BufferedOutputStream( outputStream ) ;

            byte[] b=new byte[1024];   //代表一次最多读取1KB的内容

            int length = 0 ; //代表实际读取的字节数
            while( (length = bufferedInputStream.read( b ) )!= -1 ){
                //length 代表实际读取的字节数
                bufferedOutputStream.write(b, 0, length );
            }
            //缓冲区的内容写入到文件
            bufferedOutputStream.flush();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }catch (IOException e) {
            e.printStackTrace();
```

```
        }finally {

            if( bufferedOutputStream != null ){
                try {
                    bufferedOutputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

            if( bufferedInputStream != null){
                try {
                    bufferedInputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

            if( inputStream != null ){
                try {
                    inputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

            if ( outputStream != null ) {
                try {
                    outputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

        }
    }
}
```

效果图：



如何正确的关闭流

在上面的代码中，我们关闭流的代码是这样写的。

```
finally {

            if( bufferedOutputStream != null ){
                try {
                    bufferedOutputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

            if( bufferedInputStream != null){
                try {
                    bufferedInputStream.close();
                } catch (IOException e) {
```

```
                    e.printStackTrace();
                }
            }

            if( inputStream != null ){
                try {
                    inputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

            if ( outputStream != null ) {
                try {
                    outputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

        }
```

思考：在处理流关闭完成后，我们还需要关闭节点流吗？

让我们带着问题去看源码：

- bufferedOutputStream.close();

```
  /**
    * Closes this input stream and releases any system resources
    * associated with the stream.
    * Once the stream has been closed, further read(), available(), reset(),
    * or skip() invocations will throw an IOException.
    * Closing a previously closed stream has no effect.
    *
    * @exception  IOException  if an I/O error occurs.
    */
   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()
       }
   }
```

close（ ）方法的作用

1、关闭输入流，并且释放系统资源

2、BufferedInputStream装饰一个 InputStream 使之具有缓冲功能，is要关闭只需要调用最终被装饰出的对象的close()方法即可，因为它最终会调用真正数据源对象的 close()方法。因此，可以只调用外层流的close方法关闭其装饰的内层流。

那么如果我们想逐个关闭流，我们该怎么做？

答案是：先关闭外层流，再关闭内层流。一般情况下是：先打开的后关闭，后打开的先关闭；另一种情况：看依赖关系，如果流a依赖流b，应该先关闭流a，再关闭流b。例如处理流a依赖节点流b，应该先关闭处理流a，再关闭节点流b

看懂了怎么正确的关闭流之后，那么我们就可以优化上面的代码了，只关闭外层的处理流。

```
finally {

            if( bufferedOutputStream != null ){
                try {
                    bufferedOutputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }

            if( bufferedInputStream != null){
                try {
                    bufferedInputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }

```

```
18                  }
                }
```

个人微信号： zhaoyanjun125 ， 欢迎关注

- 上一篇 Java IO流学习总结二：File
- 下一篇 Java IO流学习总结四：缓冲流-BufferedReader、BufferedWriter

Java中BufferedInputStream和BufferedOutputStream基本使用详解

BufferedInputStream的使用 BufferedInputStream:缓冲字节输入流，是一个高级流(处理流)，与其他低级流配合使用。 构造方法//创建一个 BufferedInpu...

lyb1832567496 2016年10月03日 13:26 10753

BufferedInputStream与BufferedOutputStream用法简介

BufferedInputStream是带缓冲区的输入流，默认缓冲区大小是8M，能够减少访问磁盘的次数，提高文件读取性能；BufferedOutputStream是带缓冲区的输出流，能够提高文件的写入...

liaisuo 2014年09月29日 00:10 36153

java中讲讲BufferedOutputStream的用法，举例？

1.3 BufferedOutputStream的用法 马克-to-win：BufferedOutputStream顾名思义就是它有一个内部的buffer（缓存），当写数据时，可以批量的写。它的工作...

mark_to_win 2017年05月01日 14:59 376

BufferedInputStream和BufferedOutputStream详解

这两个类分别是FilterInputStream和FilterOutputStream的子类，作为装饰器子类，使用它们可以防止每次读取/发送数据时进行实际的写操作，代表着使用缓冲区。 了解这两个类之前...

u012483425 2015年06月17日 20:20 1743

BufferedInputStream和BufferedOutputStream的滥用

BufferedInputStream和BufferedOutputStream的滥用 这里标题中的BufferedInputStream不仅仅指BufferedInputStream，Buffe...

xuesong123 2016年11月19日 01:05 1862

BufferedOutputStream的缓存功能解析（源码阅读）

要介绍BufferedOutputStream，我们先了解一下OutputStream类 抽象类OutputStream类有三个write方法 public abstract void write(...

u010149189 2016年05月09日 19:40 1524

java BufferedOutputStream和BufferedInputStream

BufferedInputStream是套在某个其他的InputStream外，起着缓存的功能，用来改善里面那个InputStream的性能（如果可能的话），它自己不能脱离里面那个单独存在。FileI...

jiachangbin1989 2016年04月19日 10:58 479

java中常用io流 BufferedInputStream和BufferedOutputStream

BufferedInputStream 为另一个输入流添加一些功能，即缓冲输入以及支持 mark 和 reset 方法的能力。在创建 BufferedInputStream 时，会创建一个内部缓冲区...

j_a_v_a_guan 2015年08月21日 19:32 498

Java-IO之BufferedOutputStream（缓冲输出流）

BufferedOutputStream是缓冲输出流，继承于FilterOutputStream，作用是为另外一个输出流提供换从功能。 主要函数列表：BufferedOutputStream(Out...

qq924862077 2016年10月03日 19:55 2232

DataOutputStream 类与BufferedOutputStream类的区别是什么

DataOutputStream 类与BufferedOutputStream类的区别是什么DataOutputStream dataout=new DataOutputStream(new F...

silentJesse 2007年12月14日 13:09 4759

BufferedInputStream和BufferedOutputStream用法 解决乱码

BufferedInputStream和BufferedOutputStream是过滤流,需要使用已存在的节点来构造,即必须先有InputStream或OutputStream,相对直接读写,这两个流...

hustwht 2016年07月10日 10:17 836

BufferedOutputStream 介绍_动力节点Java学院整理

2017年10月30日 16:19 107KB 下载

FileOutputStream、BufferedOutputStream、FileWriter三种文件写入的对比

因为需要把在线的apk文件下载到本地，一上来就使用了FileWriter的方式进行文件写入，结果发现下载到本地的apk会提示安装包损坏，经过排查，原来FileWriter是使用的字符写入的方式，而可执...

Anlegor 2015年03月06日 16:29 7221

对OutputStream类的flush()方法的误解

beinlife 2016年09月27日 10:38 4302

最近使用java的FileOutputStream写文件，调用到了flush()方法。 于是查看了FileInputStream类的源代码，发现flush()其实是继承于其父类OutputStrea...

Java中的高级I/O流-缓冲流、数据流以及对象流

BlueSky_USC 2016年11月28日 22:25 1824

Java中的高级I/O流-缓冲流、数据流以及对象流 前言：通过前面的学习，已经学完了Java中的基本流；Java中的流有字节流和字符流两大类，而每一种流都有对应的输入和输出流； 1、字节流 1...

java压缩文件

tiger925 2013年02月15日 11:27 133

Java代码 /** * 压缩文件 * * @method zip_file * @param file_path 需压缩的文件路径...

IO流之缓冲流


u014467099 2015年04月20日 17:41 170

缓冲流本身不具IO功能，只是在别的流上加上缓冲提高效率，像是为别的流装上一种包装。当对文件或其他目标频繁读写或操作效率低，效能差。这时使用缓冲流能够更高效的读写信息。因为缓冲流先将数据缓存起来，然后一...

java IO流之三 使用缓冲流来读写文件


haluoluo211 2016年08月11日 15:39 5766

http://blog.csdn.net/a107494639/article/details/7586689 一、通过BufferedReader和BufferedWriter来读写文件 ...

恭喜：一个公式教你秒懂天下英语

老司机教你一个数学公式秒懂天下英语

【Java】缓冲流如何提高性能


reliveIT 2015年05月18日 12:32 2017

缓冲流如何提高性能（随笔，写的比较粗糙，详情还是请自行观赏源码）

Java 缓冲流简介及简单用法


nvd11 2014年07月01日 23:55 3484

在java编程中, 我们有时会tingda