笔记总链接：http://bbs.itheima.com/thread-200600-1-1.html

8、IO

8.2 IO流

8.2.2 IO流常用基类-字符流

练习：

将d盘的一个文本文件复制到d盘。

分析：

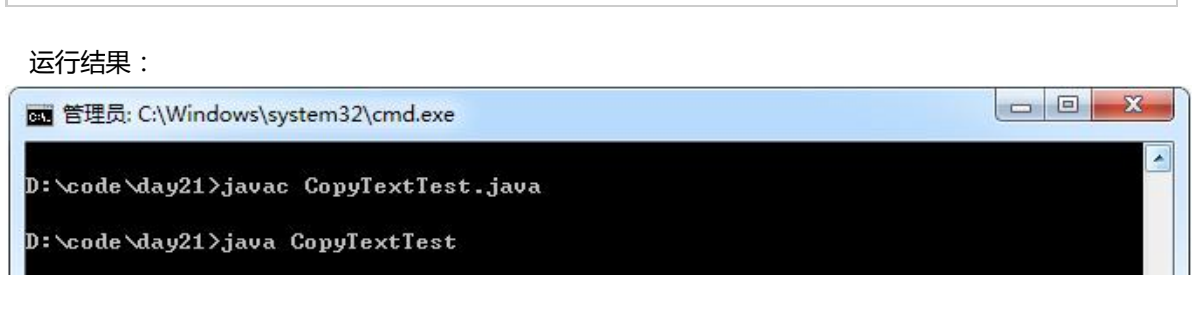
读取d盘demo.txt文件中的数据，将这些数据写入到d盘copyText_1.txt文件当中。既然是操作文本数据，使用字符流。

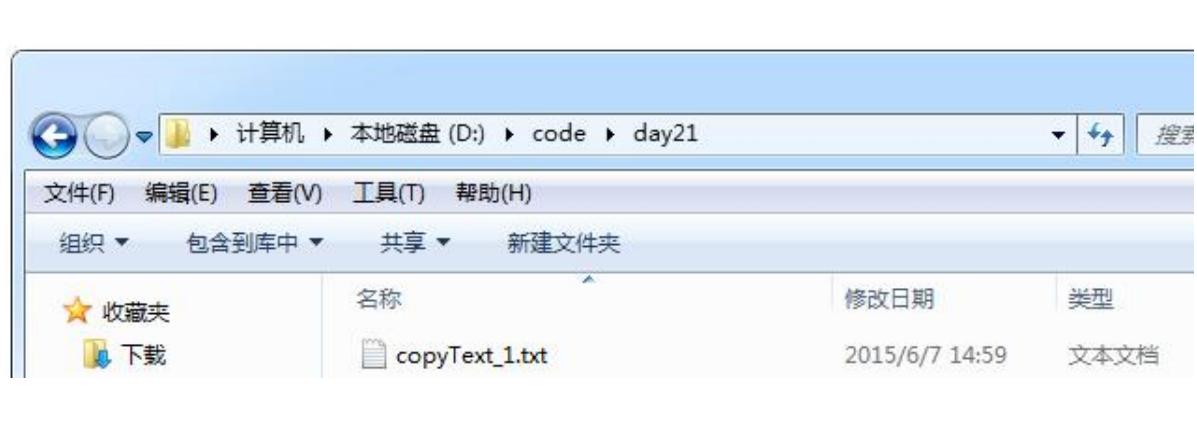
方式1：使用read()读取文本文件数据。

代码：

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class CopyTextTest{
        public static void main(String[] args) throws IOException {
              //1、读取一个已有的文本文件，使用字符读取流和文件相关联
              FileReader fr = new FileReader("demo.txt" );

              //2、创建一个目的，用于存储读到数据。
              FileWriter fw = new FileWriter("copyText_1.txt" );

              //3、频繁的读写操作
              int ch = 0;
              while((ch = fr.read()) != -1){
                   fw.write(ch);
              }

              //4、关闭字节流
              fw.close();
              fr.close();
       }
}
```
复制代码

运行结果：



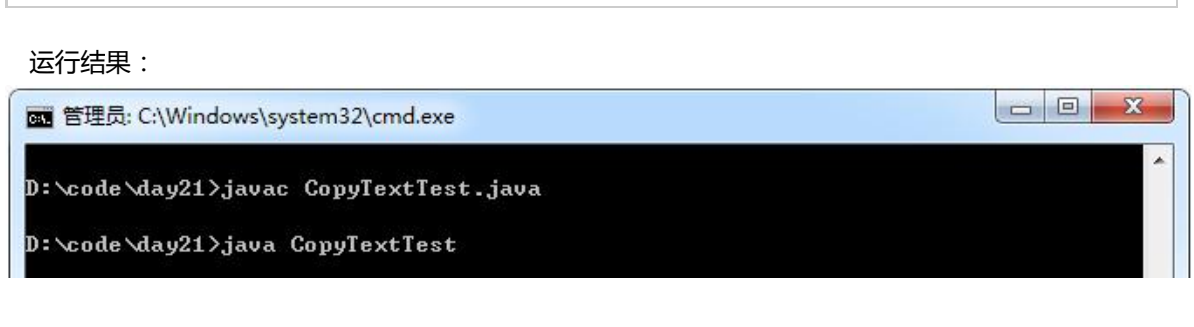


方式2：使用read(char[])读取文本文件数据。

代码：

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class CopyTextTest{
        private static final int BUFFER_SIZE = 1024;

        public static void main(String[] args){
             FileReader fr = null;
             FileWriter fw = null;

              try{
                    fr = new FileReader("demo.txt" );
                    fw = new FileWriter("copytest_2.txt" );

                     //创建一个临时容器，用于缓存读取到的字符
                     char[] buf = new char[BUFFER_SIZE];

                     //定义一个变量记录读取到的字符数（其实就是往数组里装的字符个数）
                     int len = 0;

                     while((len = fr.read(buf)) != -1){
                          fw.write(buf,0,len);
                    }
             } catch(Exception e){
                     throw new RuntimeException("读写失败!");
             } finally{
                     if(fw != null){
                          try{
                               fw.close();
                          } catch(IOException e){
                               System.out.println(e.toString());
                          }
                     }
                     if(fr != null){
                          try{
                               fw.close();
                          } catch(IOException e){
                               System.out.println(e.toString());
                          }
                     }
             }
       }
}
```
复制代码

运行结果：



字符流的缓冲区

缓冲区的出现提高了对数据的读写效率。

对应类：

BufferedWriter

BufferedReader

P.S.

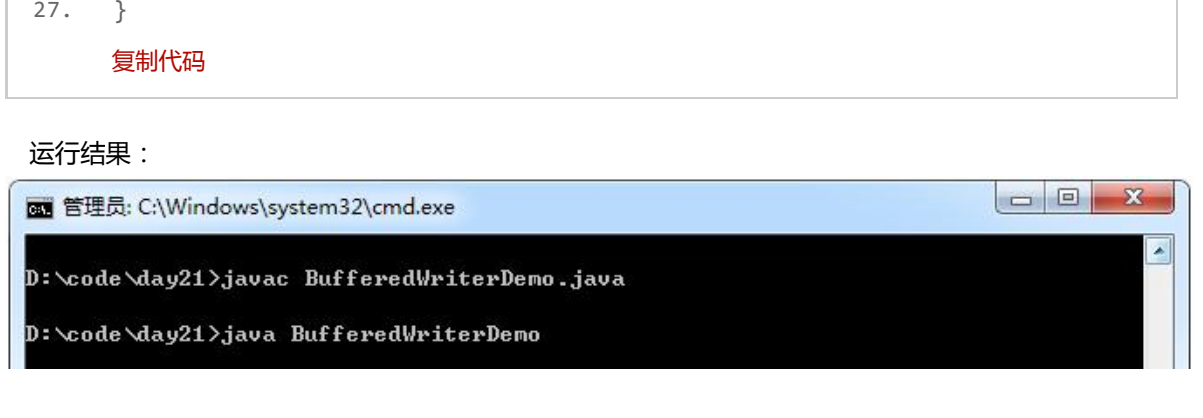
缓冲区要结合流才可以使用。

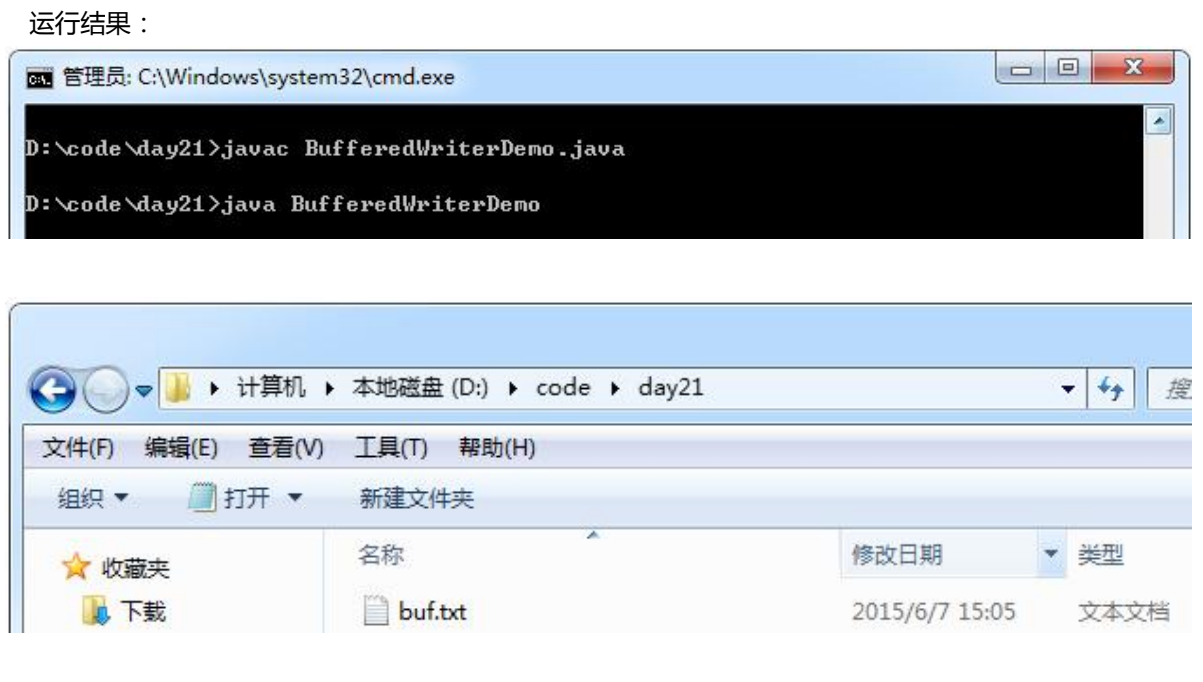
作用：在流的基础上对流的功能进行了增强。

示例6：提高写入效率，使用缓冲区。

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

public class BufferedWriterDemo{
        public static void main(String[] args) throws IOException{
             FileWriter fw = new FileWriter("buf.txt" );

              //为了提高写入的效率，使用了字符流的缓冲区
              //创建了一个字符写入流的缓冲区对象，并且指定与要被缓冲的流对象相关联
             BufferedWriter bufw = new BufferedWriter(fw);

              for(int x = 1; x <= 4; x++){
                     //使用缓冲区的写入方法将数据先写入到缓冲区中
                    bufw.write( "abcdef" + x);
                     //写入内容换行方法：newLine();
                    bufw.newLine();
                    bufw.flush();
             }

              //使用缓冲区的刷新方法将数据刷目的地中
             bufw.flush();

              //关闭缓冲区，其实关闭的就是被缓冲的流对象
             fw.close();
       }
}
```
复制代码

运行结果：

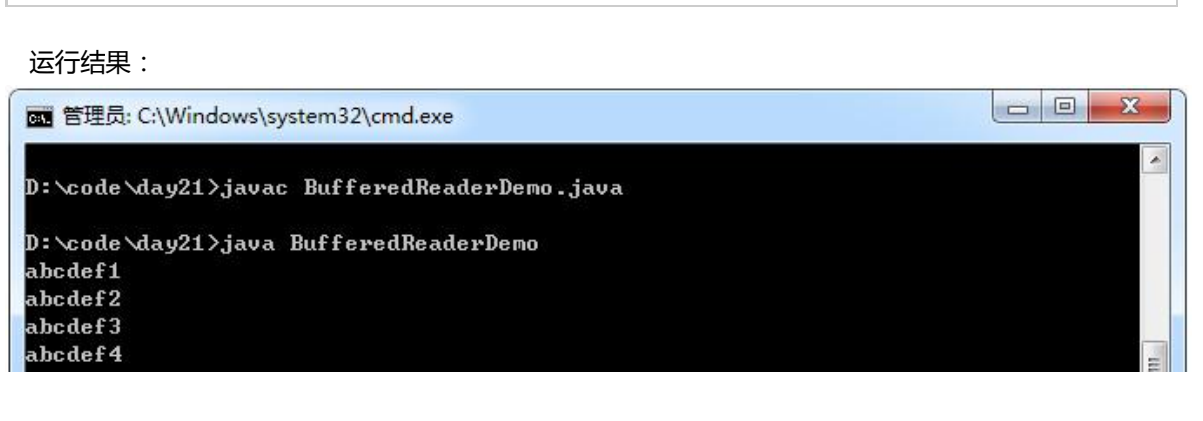




示例7：提高读取效率，使用缓冲区。

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;

public class BufferedReaderDemo{
       public static void main(String[] args) throws Exception{
            FileReader fr = new FileReader("buf.txt");

            BufferedReader bufr = new BufferedReader(fr);

            String line = null;

             while((line = bufr.readLine()) != null){
                 System.out.println(line);
            }

            bufr.close();
       }
}
```
复制代码

运行结果：



字符流缓冲区：

写入换行使用BufferedWriter类中的newLine()方法。

读取一行数据使用BufferedReader类中的readLine()方法。

bufr.read():这个read方法是从缓冲区中读取字符数据，所以覆盖了父类中的read方法。

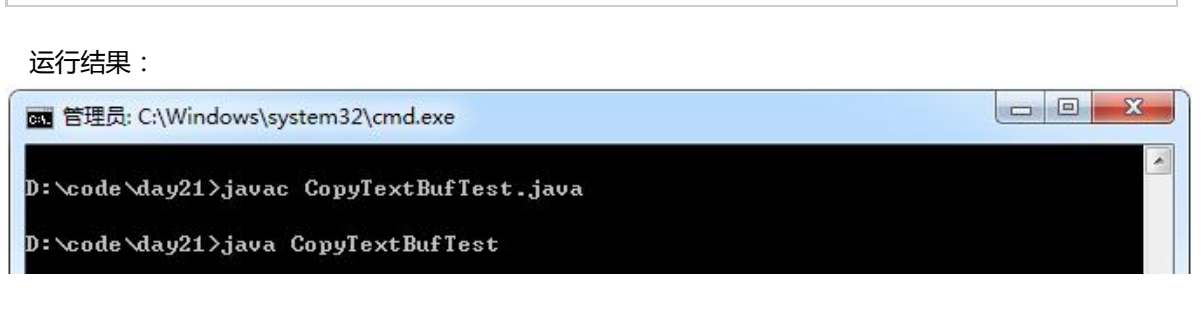
bufr.readLine():另外开辟了一个缓冲区，存储的是原缓冲区一行的数据，不包含换行符。

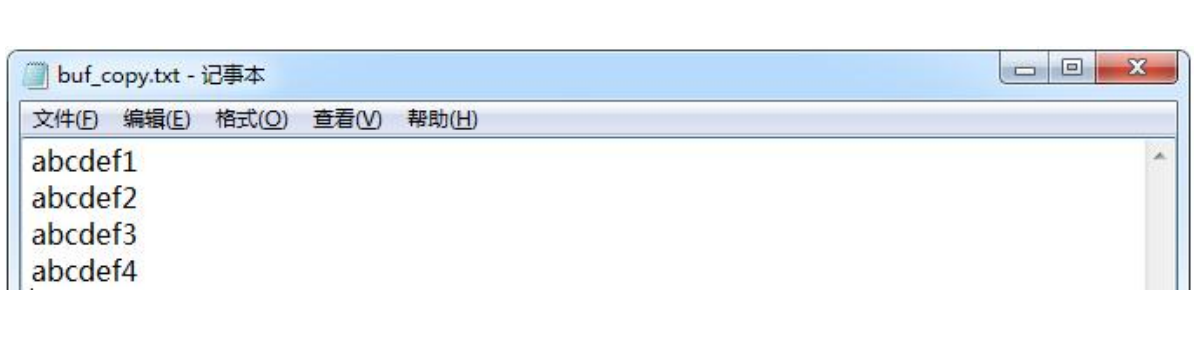
原理：使用了读取缓冲区的read方法，将读取到的字符进行缓冲并判断换行标记，将标记前的缓冲数据变成字符串返回。

示例8：

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;

public class CopyTextBufTest{
       public static void main(String[] args) throws Exception {
            FileReader fr = new FileReader("buf.txt");
            BufferedReader bufr = new BufferedReader(fr);

            FileWriter fw = new FileWriter("buf_copy.txt");
            BufferedWriter bufw = new BufferedWriter(fw);

            String line = null;

             //方式一
             while((line = bufr.readLine()) != null){
                 bufw.write(line);
                 bufw.newLine();
                 bufw.flush();
            }

             //方式二
             /*
            int ch = 0;

            while((ch = bufr.read()) != -1){
                 bufw.write(ch);
            }
            */

            bufr.close();
            bufw.close();
       }
}
```
复制代码

运行结果：





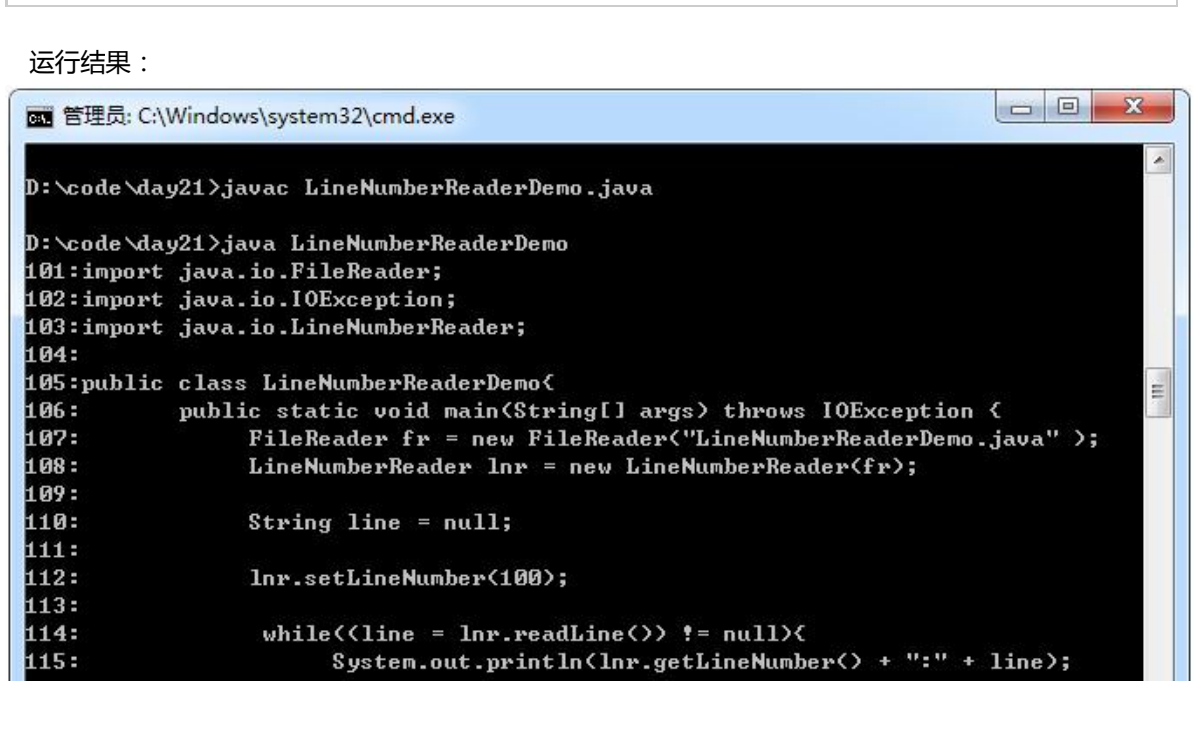
LineNumberReader

跟踪行号的缓冲字符输入流。此类定义了方法 setLineNumber(int) 和 getLineNumber()，它们可分别用于设置和获取当前行号。

示例：

```
import java.io.FileReader;
import java.io.IOException;
import java.io.LineNumberReader;

public class LineNumberReaderDemo{
        public static void main(String[] args) throws IOException {
             FileReader fr = new FileReader("LineNumberReaderDemo.java" );
             LineNumberReader lnr = new LineNumberReader(fr);

             String line = null;

             lnr.setLineNumber(100);

              while((line = lnr.readLine()) != null){
                    System.out.println(lnr.getLineNumber() + ":" + line);
             }

             lnr.close();
       }
}
```
复制代码

运行结果：



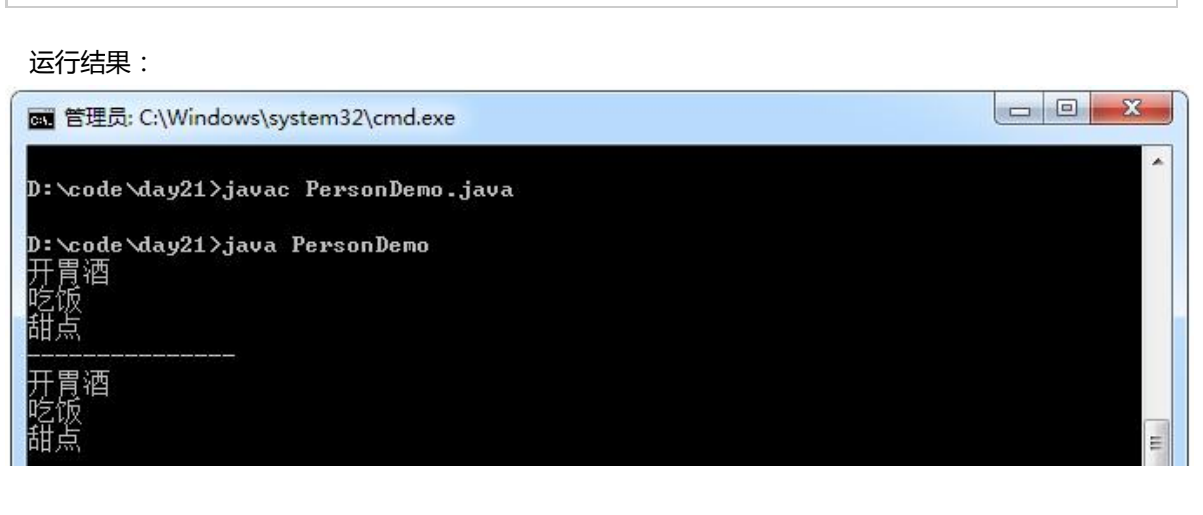
8.2.3 装饰设计模式

对原有类进行了功能的改变，增强。

示例：

```
class Person{
        void chifan(){
             System.out.println( "吃饭");
       }
}

//采用装饰的方式增强Person类
//这个类的出现是为了增强Person而出现的
class NewPerson{
        private Person p;

       NewPerson(Person p){
              this.p = p;
       }

        public void chifan(){
             System.out.println( "开胃酒");
             p.chifan();
             System.out.println( "甜点");
       }
}

//采用继承的方式增强Person类
class NewPerson2 extends Person{
        public void chifan(){
             System.out.println( "开胃酒");
              super.chifan();
             System.out.println( "甜点");
       }
}

public class PersonDemo{
        public static void main(String[] args){
             Person p = new Person();
             NewPerson np1 = new NewPerson(p);
             np1.chifan();

             System.out.println( "---------------");

             NewPerson2 np2 = new NewPerson2();
             np2.chifan();
       }
}
```
复制代码

运行结果：



装饰和继承都能实现一样的特点：进行功能的扩展增强。有什么区别呢？

首先有一个继承体系：

Writer

|--TextWriter:用于操作文本

|--MediaWriter:用于操作媒体

如果想要对操作的动作进行效率的提高，按照面向对象，可以通过继承的方式对具体的对象进行功能的扩展，那么就需要加入缓冲技术。

Writer

|--TextWriter:用于操作文本

|--BufferTextWriter:加入了缓冲技术的操作文本的对象

|--MediaWriter:用于操作媒体

|--BufferMediaWriter:加入了缓冲技术的操作媒体的对象

以上方式并不理想，如果这个体系需要再进行功能扩展，又多了更多流对象。

这样就会发现只为提高功能，导致继承体系越来越臃肿，不够灵活。

重新思考问题：

既然加入的都是同一种技术--缓冲。

前一种是让缓冲和自己的流对象相结合。

可不可以将缓冲进行单独的封装，哪个对象需要缓冲就将哪个对象和缓冲关联。

```
class Buffer {
      Buffer(TextWriter w){}
      Buffer(MediaWriter w){}
}
```
复制代码

简化为：

```

class BufferedWriter extends Writer{
      BufferedWriter(Writer w){}
}
```
复制代码

Writer

|--TextWriter:用于操作文本

|--MediaWriter:用于操作媒体

|--BufferedWriter:用于提高效率

可见：装饰比继承灵活。

特点：装饰类和被装饰类都必须所属同一个接口或者父类。

练习：

自定义一个读取缓冲区类，模拟一个BufferedReader。

分析：

缓冲区中无非就是封装了一个数组，并对外提供了更多的方法对数组进行访问，其实这些方法最终操作的都是数组的角标。

缓冲的原理：

其实就是从源中获取一批数据到缓冲区中，再从缓冲区中不断地取出一个一个数据。

在此次取完后，再从源中继续取一批数据进缓冲区，当源中的数据取完时，用-1作为结束标记。

代码：

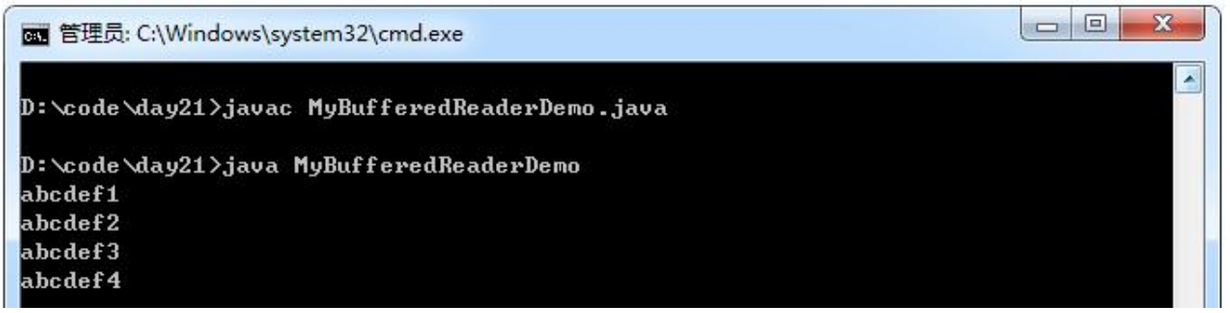
```
import java.io.FileReader;
import java.io.IOException;
import java.io.Reader;

class MyBufferedReader{

       private Reader r;

       //定义一个数组作为缓冲区
       private char[] buf = new char[1024];

       //定义一个指针用于操作这个数组中的元素，当操作到最后一个元素后，指针应该归零
       private int pos = 0;

       //定义一个计数器用于记录缓冲区中的数据个数，当该数据减到0，就从源中继续获取数据到缓冲区中
       private int count = 0;

       MyBufferedReader(Reader r){
              this .r = r;
       }

       //该方法从缓冲区中一次取一个字符
       public int myRead() throws IOException {
              //从源中获取一批数据到缓冲区中，需要先做判断，只有计数器为0时，才需要从源中获取数据
              if (count == 0){
```

```
26.                     count = r.read(buf);
27.
28.                     //每次获取数据到缓冲区后，角标归零
29.                     pos = 0;
30.               }
31.
32.               if (count < 0)
33.                     return -1;
34.
35.               char ch = buf[pos];
36.
37.               pos++;
38.               count--;
39.
40.               return ch;
41.         }
42.
43.          public String myReadLine() throws IOException {
44.               StringBuilder sb = new StringBuilder();
45.
46.               int ch = 0;
47.               while ((ch = myRead()) != -1){
48.                     if (ch == '\r' )
49.                           continue ;
50.                     if (ch == '\n' )
51.                           return sb.toString();
52.                     //将从缓冲区读到的字符，存储到缓存行数据的缓冲区中
53.                    sb.append(( char )ch);
54.               }
55.
56.               if (sb.length() != 0){
57.                     return sb.toString();
58.               }
59.               return null ;
60.         }
61.
62.          public void myClose() throws IOException {
63.               r.close();
64.         }
65.   }
66.
67.   public class MyBufferedReaderDemo{
68.          public static void main(String[] args) throws IOException {
69.               FileReader fr = new FileReader("buf.txt" );
70.
71.               MyBufferedReader bufr = new MyBufferedReader(fr);
72.
73.               String line = null ;
74.
```

```
75.                 while ((line = bufr.myReadLine()) != null){
76.                     System.out.println(line);
77.                 }
78.
79.                 bufr.myClose();
80.             }
81.     }
```

复制代码

运行结果：



8.2.4 IO流常用基类-字节流

基本操作与字符流类相同。但它不仅可以操作字符，还可以操作其他媒体文件。

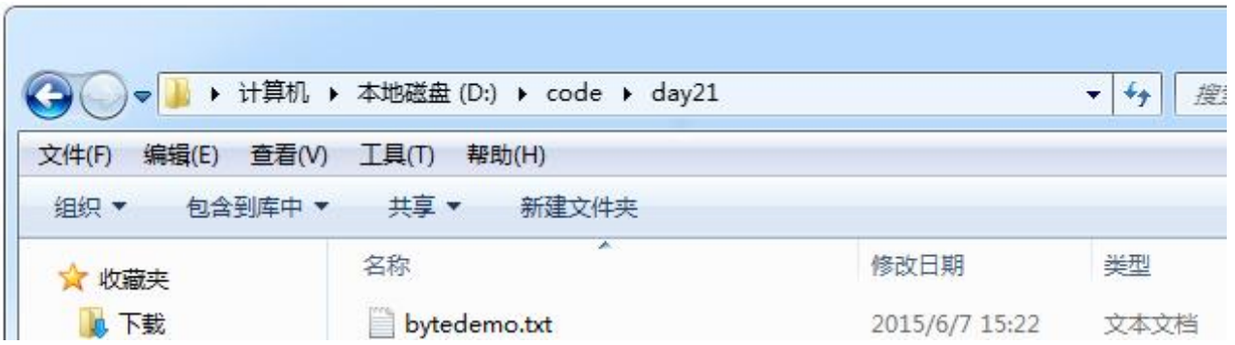
示例1：

```
01.  import java.io.FileOutputStream;
02.  import java.io.IOException;
03.
04.  public class ByteStreamDemo{
05.      public static void main(String[] args) throws IOException {
06.          demo_write();
07.      }
08.
09.      public static void demo_write() throws IOException {
10.          //1、创建字节输出流对象，用于操作文件
11.          FileOutputStream fos = new FileOutputStream( "bytedemo.txt");
12.
13.          //2、写数据，直接写入到了目的地中
14.          fos.write( "abcdefg".getBytes());
15.
16.          //关闭资源动作要完成
17.          fos.close();
18.      }
```

```
19.   }
```

复制代码

运行结果：



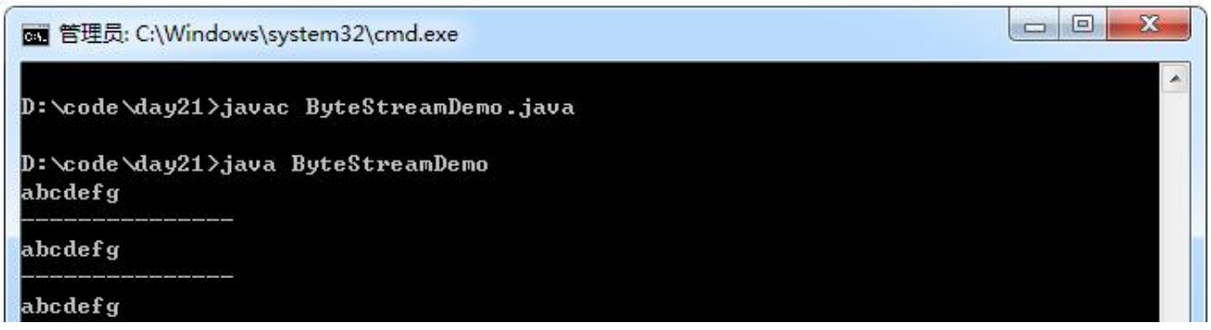
示例2：

```
import java.io.FileInputStream;
import java.io.IOException;

public class ByteStreamDemo{
        public static void main(String[] args) throws IOException {
             demo_read1();
             System.out.println( "---------------");
             demo_read2();
             System.out.println( "---------------");
             demo_read3();
        }

         //读取方式一
         public static void demo_read1() throws IOException {
               //1、创建一个读取流对象，和指定文件关联
             FileInputStream fis = new FileInputStream("bytedemo.txt" );

               //打印字符字节大小，不过要少用，文件太大，可能内存溢出
               byte[] buf = new byte[fis.available()];
             fis.read(buf);
             System.out.println( new String(buf));

             fis.close();
        }

```

```
26.          //读取方式二
27.          public static void demo_read2() throws IOException {
28.
29.                FileInputStream fis = new FileInputStream("bytedemo.txt" );
30.
31.                //建议使用这种读取数据的方式
32.                byte[] buf = new byte[1024];
33.
34.                int len = 0;
35.
36.                while((len = fis.read(buf)) != -1){
37.                     System.out.println( new String(buf,0,len));
38.               }
39.
40.               fis.close();
41.         }
42.
43.          //读取方式三
44.          public static void demo_read3() throws IOException {
45.
46.               FileInputStream fis = new FileInputStream("bytedemo.txt" );
47.
48.                //一次读取一个字节
49.                int ch = 0;
50.                while((ch = fis.read()) != -1){
51.                     System.out.print(( char)ch);
52.               }
53.
54.               fis.close();
55.         }
56.   }
```

复制代码

运行结果：



P.S.

FileOutputStream、FileInputStream的flush方法内容为空，没有任何实现，调用没有意义。

字节流的缓冲区：同样是提高了字节流的读写效率。

练习：

通过几种方式对MP3的进行拷贝，比较它们的效率。

代码：

```
01.  import java.io.BufferedInputStream;
02.  import java.io.BufferedOutputStream;
03.  import java.io.FileInputStream;
04.  import java.io.FileOutputStream;
05.  import java.io.IOException;
06.
07.  public class CopyMp3Test{
08.      public static void main(String[] args) throws IOException {
09.          copy_1();
10.          copy_2();
11.      }
12.
13.      public static void copy_1() throws IOException {
14.          FileInputStream fis = new FileInputStream("0.mp3" );
15.          FileOutputStream fos = new FileOutputStream("1.mp3" );
16.
17.           byte[] buf = new byte[1024];
18.
19.           int len = 0;
20.
21.           while((len = fis.read(buf)) != -1){
22.               fos.write(buf,0,len);
23.          }
24.
25.          fis.close();
26.          fos.close();
27.      }
28.
29.      public static void copy_2() throws IOException {
30.          FileInputStream fis = new FileInputStream("0.mp3" );
31.          BufferedInputStream bufis = new BufferedInputStream(fis);
32.
33.          FileOutputStream fos = new FileOutputStream("2.mp3" );
34.          BufferedOutputStream bufos = new BufferedOutputStream(fos);
35.
```

```
36.             int ch = 0;
37.
38.             while((ch = bufis.read()) != -1){
39.                 bufos.write(ch);
40.             }
41.
42.             bufis.close();
43.             bufos.close();
44.         }
45.     }
```

复制代码

运行结果：

~END~

~爱上海，爱黑马~


Love
Shanghai
AURORA
Expo