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

8、IO

8.2 IO流

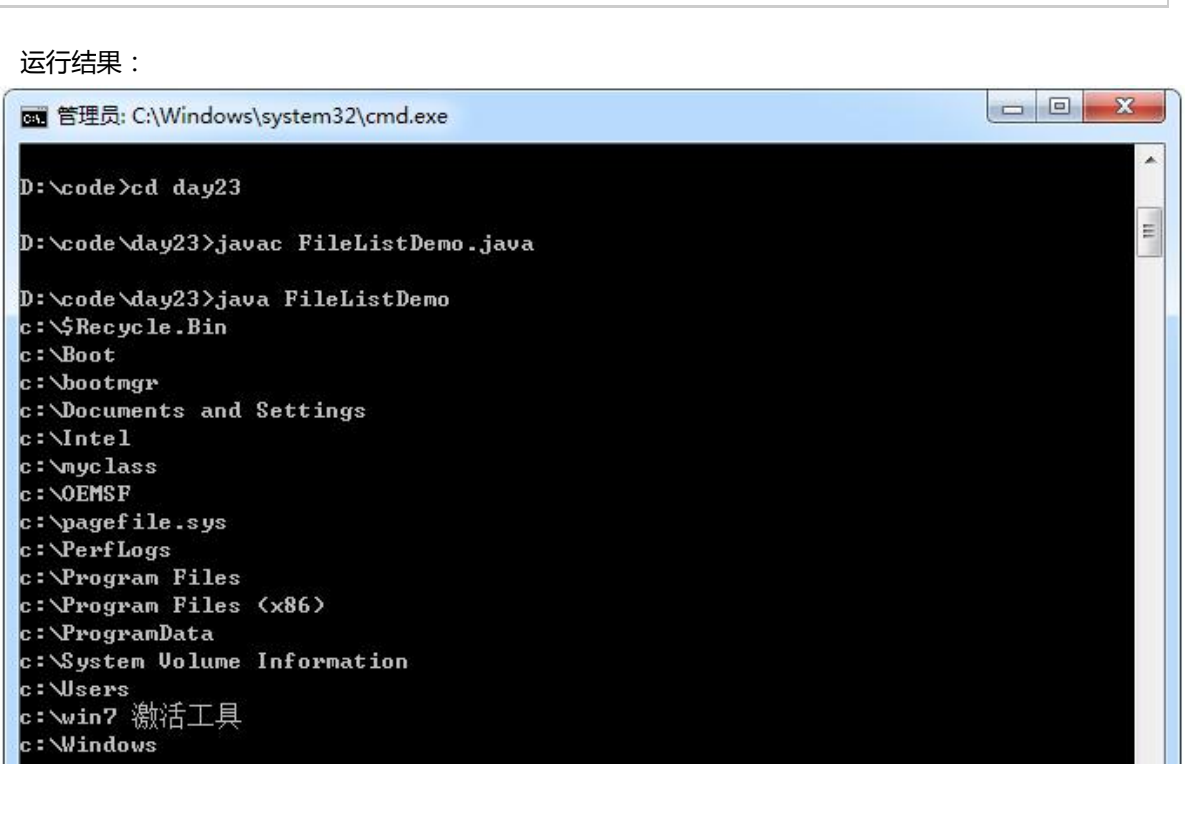
8.2.7 File类

需求：获取c盘目录下的隐藏文件。

代码：

```
import java.io.File;
import java.io.FilenameFilter;

public class FileListDemo{
      public static void main(String[] args){
           listDemo();
     }

      public static void listDemo(){
           File dir = new File("c:\\" );

           File[] files = dir.listFiles( new FilterByHidden());

            for(File file : files){
                 System.out.println(file);
           }
     }
}

class FilterByHidden implements FilenameFilter{
      public boolean accept(File dir,String name){
            return dir.isHidden();
     }
}
```
复制代码

运行结果：



8.2.8 递归

函数自身直接或者间接的调用到了自身。

一个功能在被重复使用，并每次使用时，参与运算的结果和上一次调用有关。这时可以用递归来解决问题。

P.S.

1、递归一定明确条件，否则容易栈溢出。

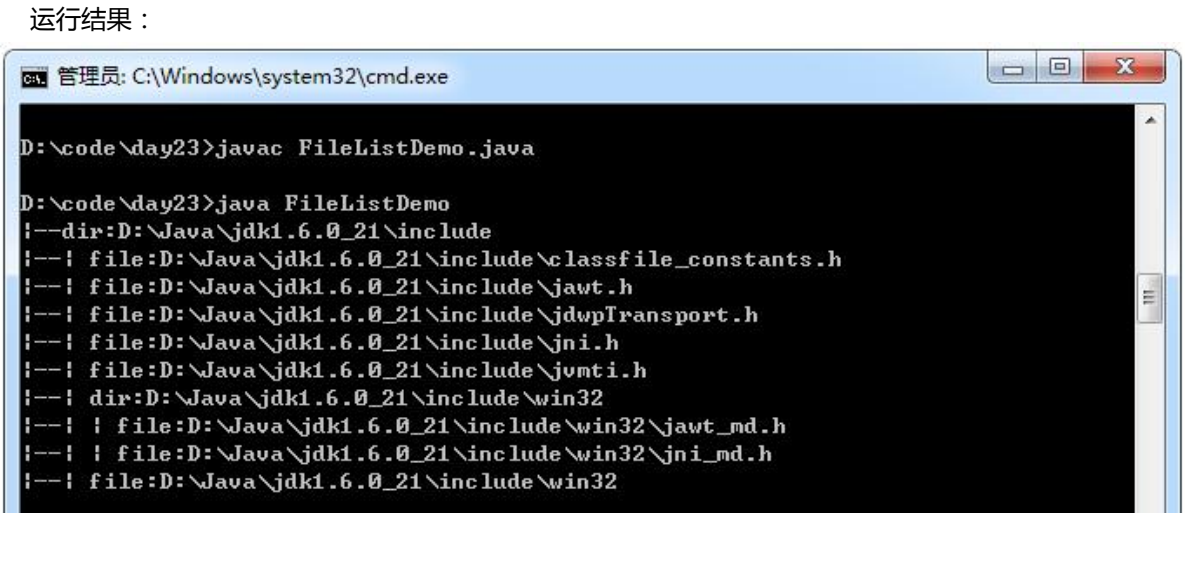
2、注意一下递归的次数。

需求：对指定目录进行所有内容的列出（包含子目录中的内容），也可以理解为深度遍历。

代码：

```
import java.io.File;

public class FileListDemo{
       public static void main(String[] args){
            File dir = new File("D:\\Java\\jdk1.6.0_02\\include" );
            listAll(dir,0);
      }

       public static void listAll(File dir, int level){
            System.out.println(getSpace(level) + "dir:" + dir.getAbsolutePath());
             //获取指定目录下当前的所有文件夹或者文件对象
            level++;
            File[] files = dir.listFiles();

             for (int x = 0; x < files.length; x++){
                   if (files[x].isDirectory()){
                        listAll(files[x],level);
                  }
                  System.out.println(getSpace(level) + "file:" + files[x].getAbsolutePath());
            }
      }

       private static String getSpace( int level){
            StringBuilder sb = new StringBuilder();

            sb.append( "|--" );
             for (int x = 0; x < level; x++){
                  sb.append( "| " );
            }

             return sb.toString();
      }
}
```
复制代码

运行结果：

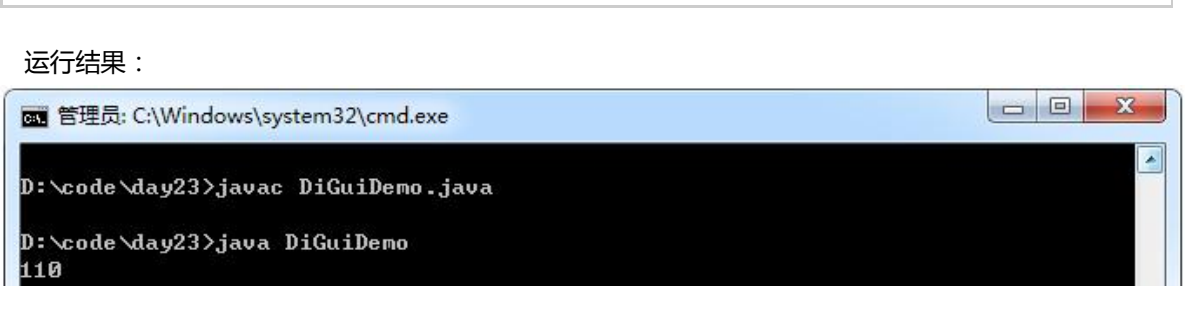


需求：利用递归求6的二进制值。

代码：

```
public class DiGuiDemo{
      public static void main(String[] args){
           toBin(6);
     }

      public static void toBin(int num){
            if(num > 0){
                 toBin(num/2);
                 System.out.print(num%2);
           }
     }
}
```
复制代码

运行结果：

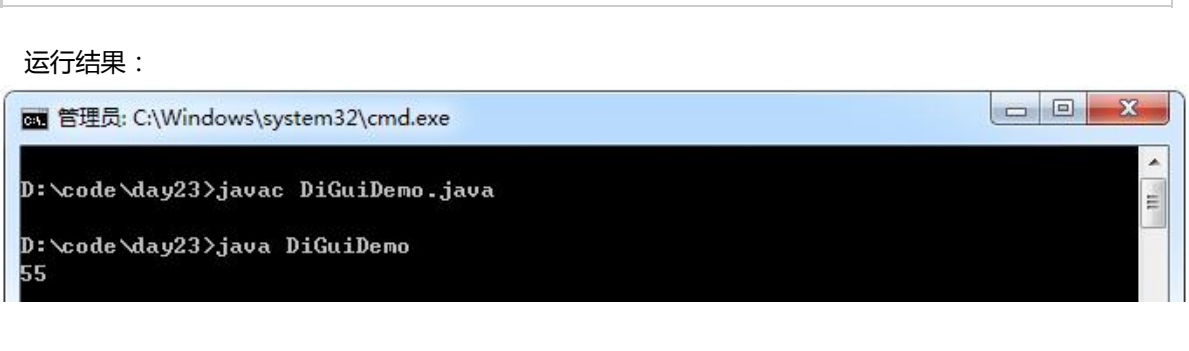


需求：利用递归求1到10的和。

代码：

```
public class DiGuiDemo{
      public static void main(String[] args){
            int sum = getSum(10);
           System.out.println(sum);
     }

      public static int getSum(int num){
            if(num == 1)
                  return 1;
            return num + getSum(num - 1);
     }
}
```
复制代码

运行结果：



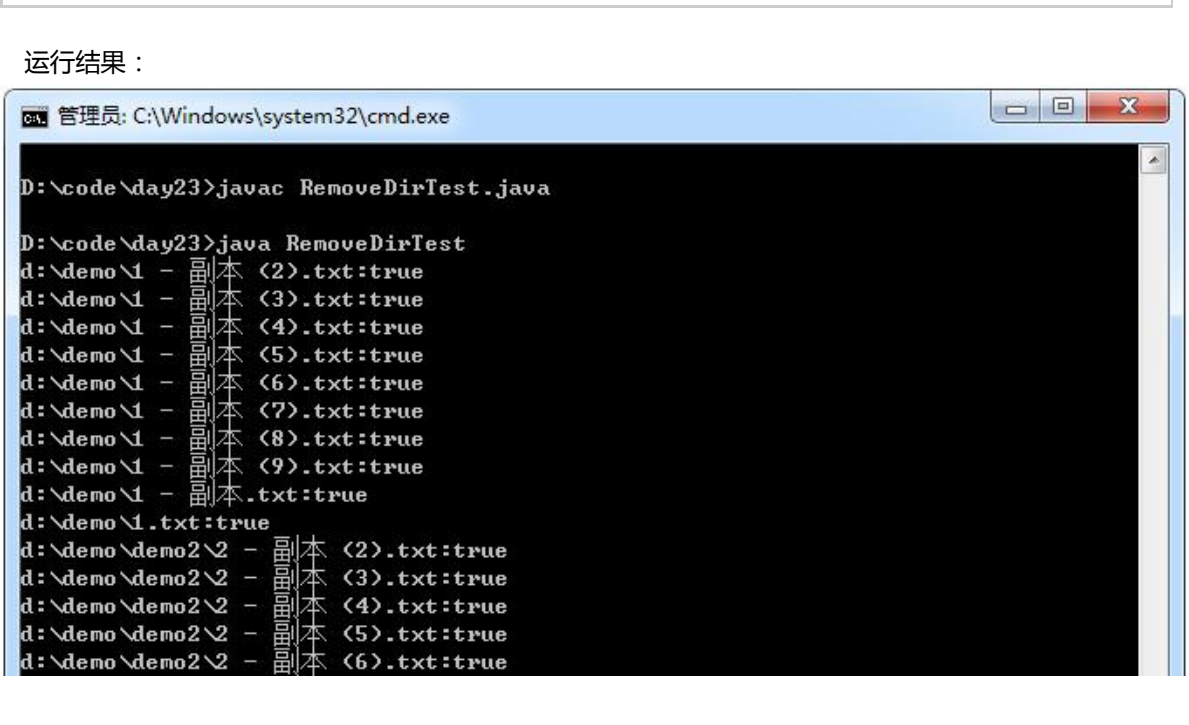
需求：删除一个带内容的目录。

原理：必须从里面往外删，需要进行深度遍历。

代码：

```
import java.io.File;

public class RemoveDirTest{
      public static void main(String[] args){
           File dir = new File("d:\\demo" );
           removeDir(dir);
     }

      public static void removeDir(File dir){
           File[] files = dir.listFiles();

            for(File file : files){
                  if(file.isDirectory()){
                       removeDir(file);
                 } else{
                       System.out.println(file + ":" + file.delete());
                 }
           }
           System.out.println(dir + ":" + dir.delete());
     }
}
```
复制代码

运行结果：



8.2.9 Properties集合

Map

|--Hashtable

|--Properties

特点：

1. 该集合中的键和值都是字符串类型。

2. 集合中的数据可以保存到流中，或者从流中获取。

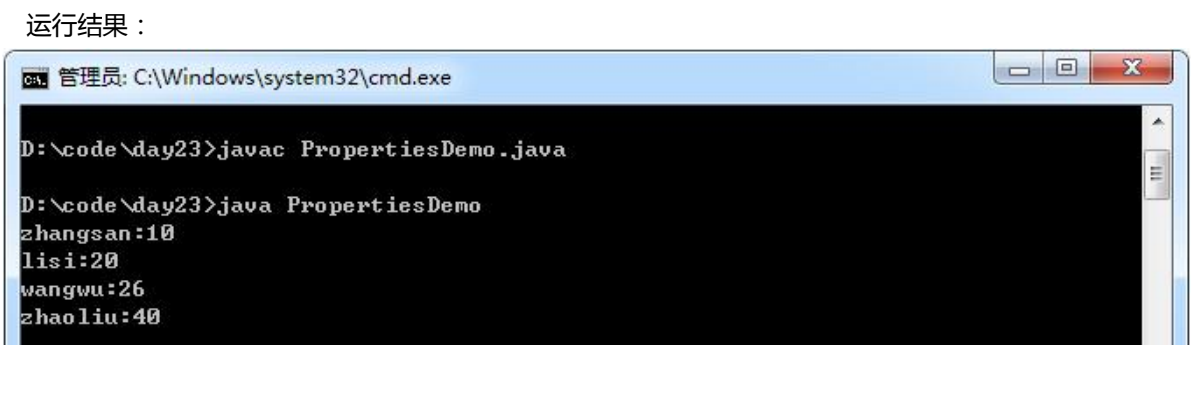
3. 通常该集合用于操作以键值对形式存在的配置文件。

Properties集合的存和取。

示例1：

```
import java.util.Properties;
import java.util.Set;

public class PropertiesDemo{
      public static void main(String[] args){
           propertiesDemo();
     }

      public static void propertiesDemo(){
            //创建一个Properties集合
           Properties prop = new Properties();

            //存储元素
           prop.setProperty( "zhangsan","10" );
           prop.setProperty( "lisi","20" );
           prop.setProperty( "wangwu","30" );
           prop.setProperty( "zhaoliu","40" );

            //修改元素
           prop.setProperty( "wangwu","26" );

            //取出所有元素
           Set<String> names = prop.stringPropertyNames();

            for(String name : names){
                 String value = prop.getProperty(name);
                 System.out.println(name + ":" + value);
           }
     }
}
```
复制代码

运行结果：

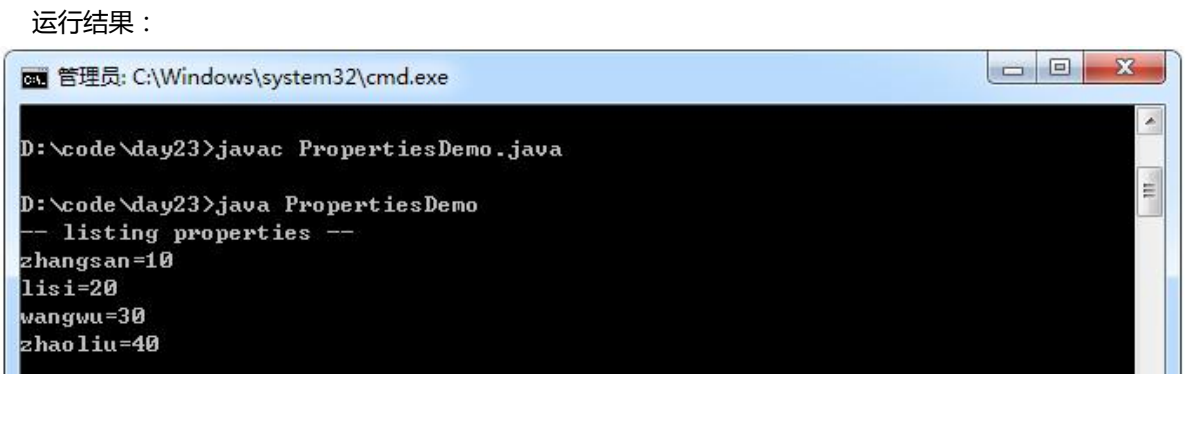


演示Properties集合和流对象相结合的功能。

示例2：

```
import java.util.Properties;

public class PropertiesDemo{
       public static void main(String[] args){
            propertiesDemo();
      }

       public static void propertiesDemo(){
            Properties prop = new Properties();

            prop.setProperty( "zhangsan","10" );
            prop.setProperty( "lisi","20" );
            prop.setProperty( "wangwu","30" );
            prop.setProperty( "zhaoliu","40" );

            prop.list(System.out);
      }
}
```
复制代码

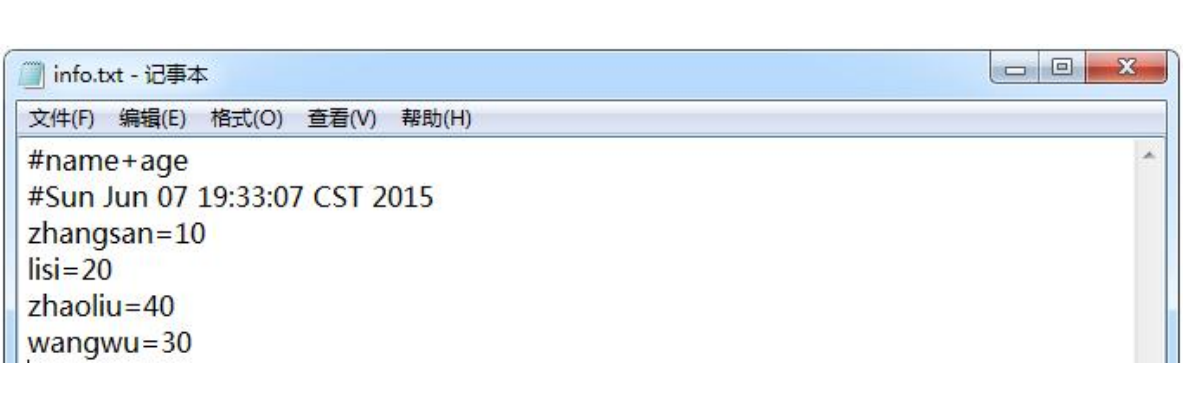
运行结果：



示例3：

```
import java.io.FileOutputStream;
import java.util.Properties;

public class PropertiesDemo{
       public static void main(String[] args) throws Exception  {
            propertiesDemo();
      }

       public static void propertiesDemo() throws Exception {
            Properties prop = new Properties();

            prop.setProperty( "zhangsan","10" );
            prop.setProperty( "lisi","20" );
            prop.setProperty( "wangwu","30" );
            prop.setProperty( "zhaoliu","40" );

             //想要将这些集合中的字符串键值信息持久化存储到文件中
             //需要关联输出流
            FileOutputStream fos = new FileOutputStream("info.txt" );

             //将集合中数据存储到文件中
            prop.store(fos, "name+age");

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

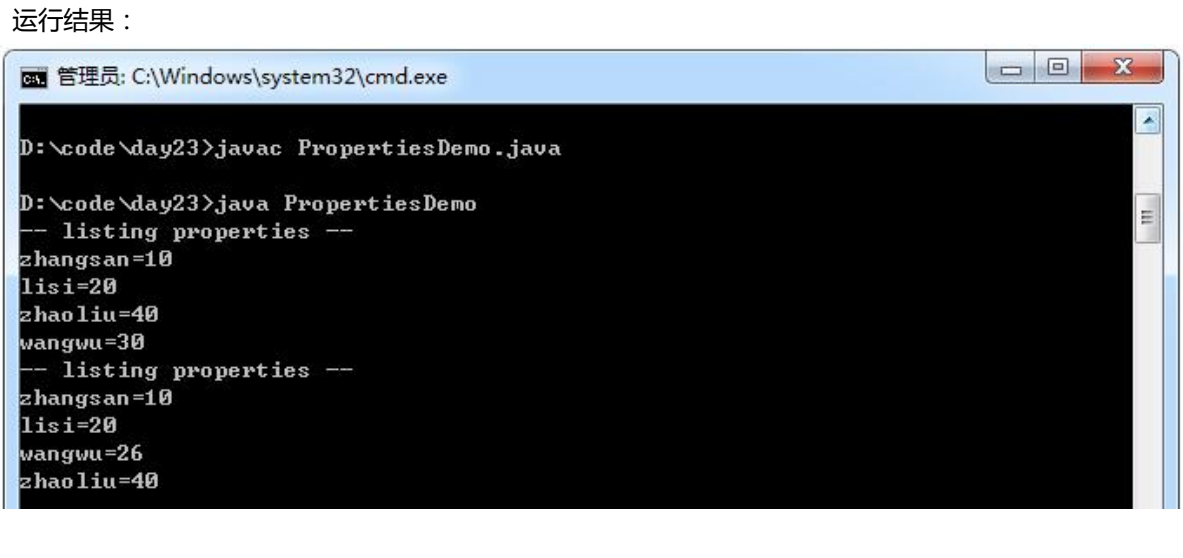
运行结果：



示例4：

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Properties;

public class PropertiesDemo{
       public static void main(String[] args) throws Exception {
            propertiesDemo();
      }

       public static void propertiesDemo() throws Exception {
            Properties prop = new Properties();

             //集合中的数据来自于一个文件。
             //注意：必须要保证该文件中的数据是键值对。
             //需要使用到读取流
            FileInputStream fis = new FileInputStream("info.txt" );

             //使用load方法
            prop.load(fis);
            prop.list(System.out);

            test();
            myLoad();
      }

       //对已有的配置文件中的信息进行修改。
       //读取这个文件。并将这个文件中的键值数据存储到集合中。再通过集合对数据进行修改。
       //再通过流将修改后的数据存储到文件中
       public static void test() throws Exception {
             //读取这个文件
            File file = new File("info.txt" );
             if(!file.exists()){
                  file.createNewFile();
            }
            FileReader fr = new FileReader("info.txt" );

             //创建集合存储配置信息
            Properties prop = new Properties();

             //将流中信息存储到集合中
            prop.load(fr);

            prop.setProperty( "wangwu","26" );

            FileWriter fw = new FileWriter(file);

            prop.store(fw, "");

            fr.close();
      }

       //模拟一下load方法
       public static void myLoad() throws Exception {
            Properties prop = new Properties();

            BufferedReader bufr = new BufferedReader(new FileReader("info.txt" ));

            String line = null;

             while((line = bufr.readLine()) != null){
                   if(line.startsWith("#" ))
                         continue;
                  String[] arr = line.split( "=");
                  prop.setProperty(arr[0],arr[1]);
            }

            prop.list(System.out);
            bufr.close();
      }
}
```
复制代码

运行结果：



需求：获取一个应用程序运行的次数，如果超过5次，给出使用次数已到请注册的提示，并不要再运行程序。

思路：

1、应该有计数器

每次程序启动都需要计数一次，并且是在原有的次数上进行计数。

2、计数器就是一个变量。突然冒出一想法，程序启动时进行计数，计数器必须存在于内存并进行运算。

可是程序一结束，计数器消失了。那么再次启动该程序，计数器又重新被初始化了。

而我们需要多次启动同一个应用程序，使用的是同一个计数器。

这就需要计数器的生命周期边长，从内存存储到硬盘文件中。

3、如何使用这个计数器呢？

首先，程序启动时，应该先读取这个用于记录计数器信息的配置文件。

获取上一次计数器次数。并进行使用次数的判断。

其次，对该次数进行自增，并自增后的次数重新存储到配置文件中。

4、文件中的信息该如何进行存储并体现。

直接存储次数值可以，但是不明确该数据的含义。所以起名字就变得很重要。

这就有了名字和值的对应，所以可以使用键值对。

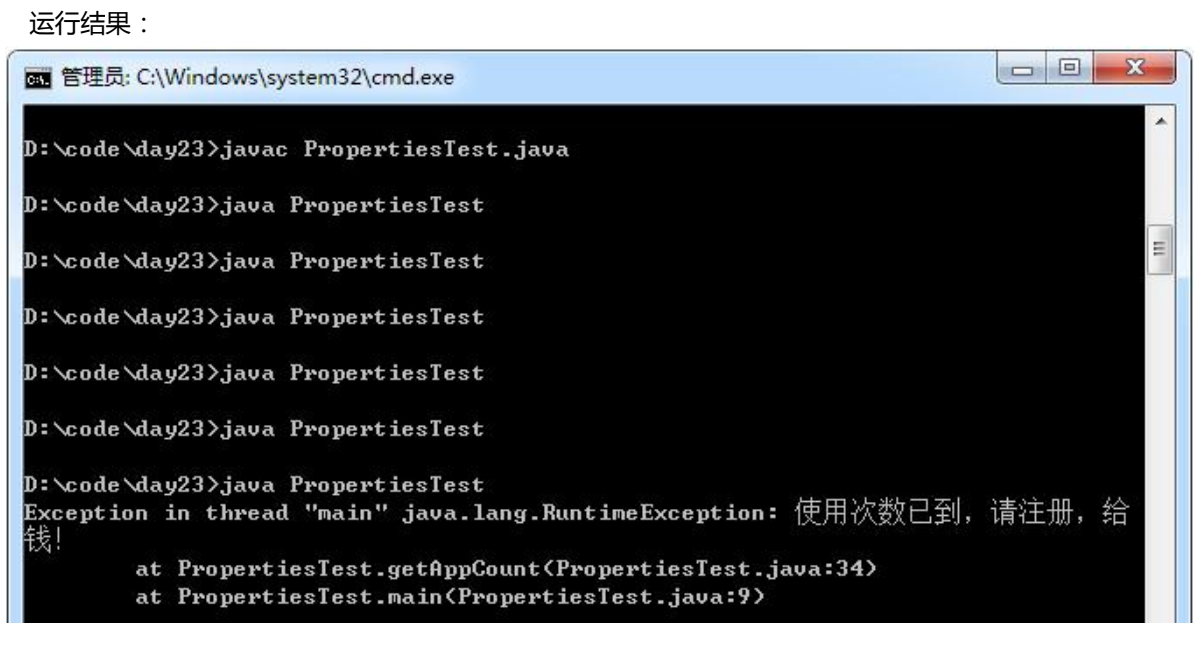
可是映射关系map集合搞定，又需要读取硬盘上的数据，所以map+io=properties。

代码：

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Properties;

public class PropertiesTest{
      public static void main(String[] args) throws IOException{
           getAppCount();
     }

      public static void getAppCount() throws IOException {
            //将配置文件封装成File对象
           File confile = new File("count.properties" );
```

```

        if(!confile.exists()){
            confile.createNewFile();
        }

        FileInputStream fis = new FileInputStream(confile);

        Properties prop = new Properties();

        prop.load(fis);

        //从集合中通过键获取次数
        String value = prop.getProperty( "time");

        //定义计数器，记录获取到的次数
        int count = 0;
        if(value != null){
            count = Integer.parseInt(value);
            if(count >= 5){
                throw new RuntimeException("使用次数已到，请注册，给钱！" );
            }
        }
        count++;

        //将改变后的次数重新存储到集合中。
        prop.setProperty( "time",count + "" );

        FileOutputStream fos = new FileOutputStream(confile);

        prop.store(fos, "");

        fos.close();
        fis.close();
    }
}
```
复制代码

运行结果：



需求：获取指定目录下，指定扩展名的文件（包含子目录中的），并且将这些文件的绝对路径写入到一个文本文件中。

简单说：就是建立一个指定扩展名的文件的列表。

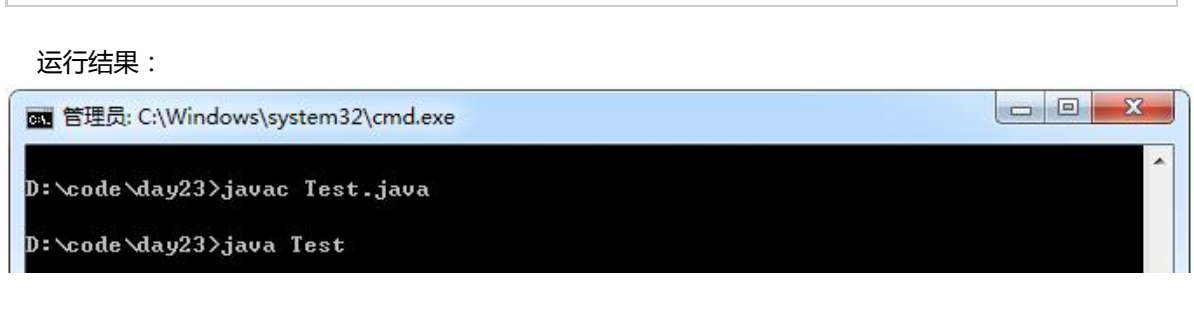
思路：

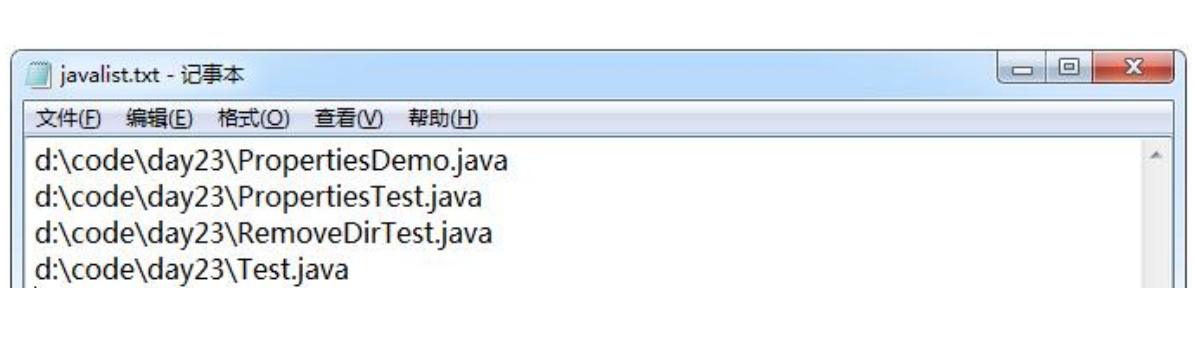
1. 必须进行深度遍历。
2. 要在遍历的过程中进行过滤，将符合条件的内容都存储到容器中。
3. 对容器中的内容进行遍历并将绝对路径写入到文件中。

代码：

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.FilenameFilter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class Test{
       public static void main(String[] args) throws IOException {
            File dir = new File("d:\\code" );
            FilenameFilter filter = new FilenameFilter(){
                   public boolean accept(File dir,String name){
                         return name.endsWith(".java" );
                   }
            };

            List<File> list = new ArrayList<File>();
            getFiles(dir,filter,list);
            File destFile = new File(dir,"javalist.txt" );
            write2File(list,destFile);
       }
        /*
       对指定目录中的内容进行深度遍历，并按照指定过滤器，进行过滤。
       将过滤后的内容存储到指定容器List中。
       */
        public static void getFiles(File dir,FilenameFilter filter,List<File> list){
            File[] files = dir.listFiles();

             for(File file : files){
                   //递归
                   if(file.isDirectory()){
                        getFiles(file,filter,list);
                  } else{
                         //对便利到的文件进行过滤器的过滤。将符合条件File对象，存储到List集合中
                         if(filter.accept(dir,file.getName())){
                              list.add(file);
                        }
                  }
            }
       }

        public static void write2File(List<File> list,File destFile) throws IOException{
            BufferedWriter bufw = null;
             try{
                  bufw = new BufferedWriter(new FileWriter(destFile));

                   for(File file : list){
                        bufw.write(file.getAbsolutePath());
                        bufw.newLine();
                        bufw.flush();
                  }
            } finally{
                   if(bufw!=null)
                         try{
                              bufw.close();
                        } catch(IOException e){
                               throw new RuntimeException("关闭失败");
                        }
            }
       }
}
```
复制代码

运行结果：





8.2.10 IO包中的其他类

打印流

PrintWriter与PrintStream：可以直接操作输入流和文件。

PrintStream为其他输出流添加了功能，使它们能够方便地打印各种数据值表示形式。

与其他输出流不同，PrintStream永远不会抛出IOException。

PrintStream打印的所有字符都使用平台的默认字符编码转换为字节。

在需要写入字符而不是写入字节的情况下，应该使用PrintWriter类。

PrintStream:

1. 提供了打印方法可以对多种数据类型值进行打印，并保持数据的表示形式
2. 它不抛IOException

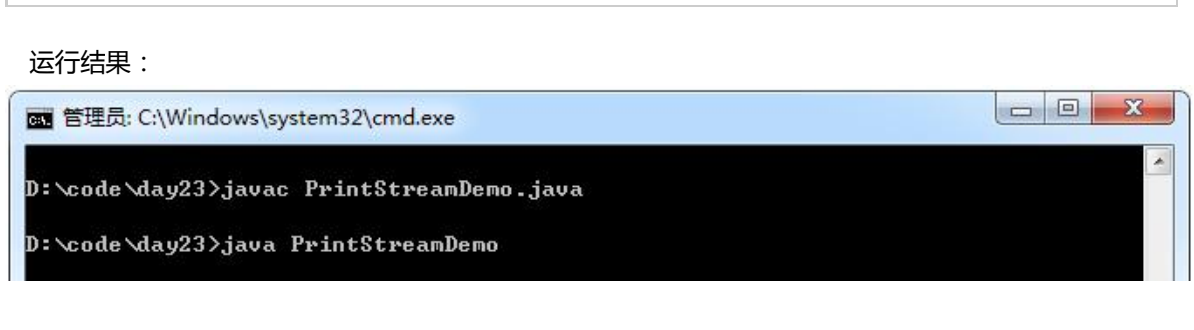
构造函数，接收三种类型的值：

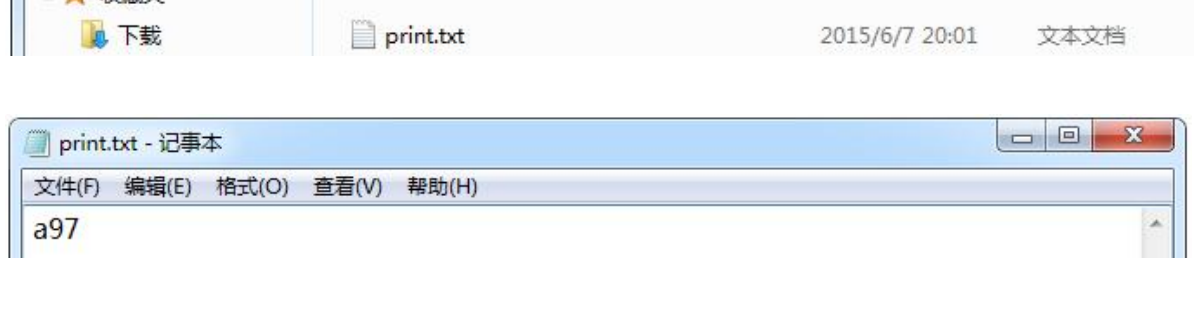
1. 字符串路径
2. File对象
3. 字节输出流

示例1：

```
import java.io.PrintStream;

public class PrintStreamDemo{
    public static void main(String[] args) throws Exception {
        PrintStream out = new PrintStream("print.txt" );

        //write(int b)方法只写最低8位
        out.write(97); //a
        //print方法将97先变成字符串保持原样将数据打印到目的地
        out.print(97); //97

        out.close();
    }
}
```
复制代码

运行结果：





PrintWriter：字符打印流

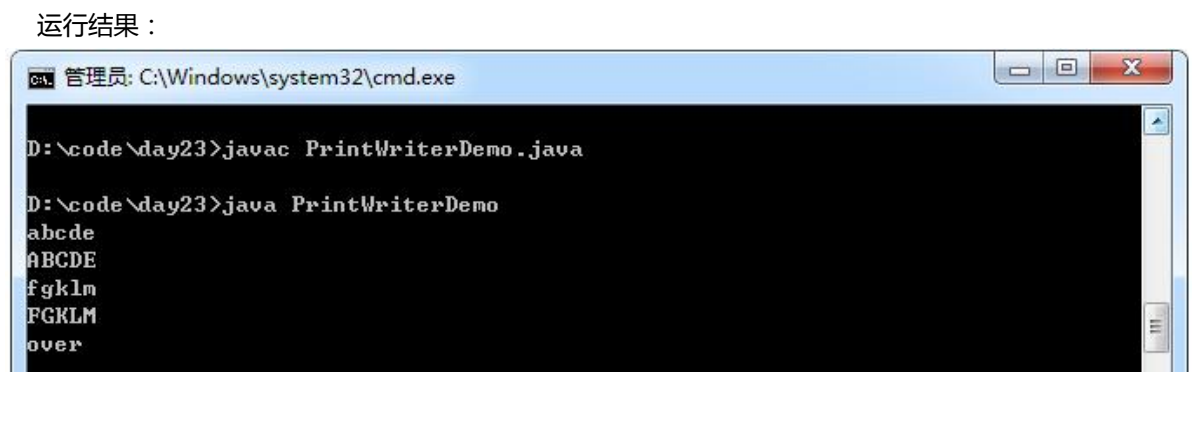
构造函数参数：

1. 字符串路径
2. File对象
3. 字节输出流
4. 字符输出流

示例2：

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.io.IOException;

public class PrintWriterDemo{
       public static void main(String[] args) throws IOException {
            BufferedReader bufr = new BufferedReader(new InputStreamReader(System.in));
            PrintWriter out = new PrintWriter(System.out);

            String line = null;
             while((line = bufr.readLine()) != null){
                   if("over" .equals(line))
                         break;
                  out.println(line.toUpperCase());
                  out.flush();
            }

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

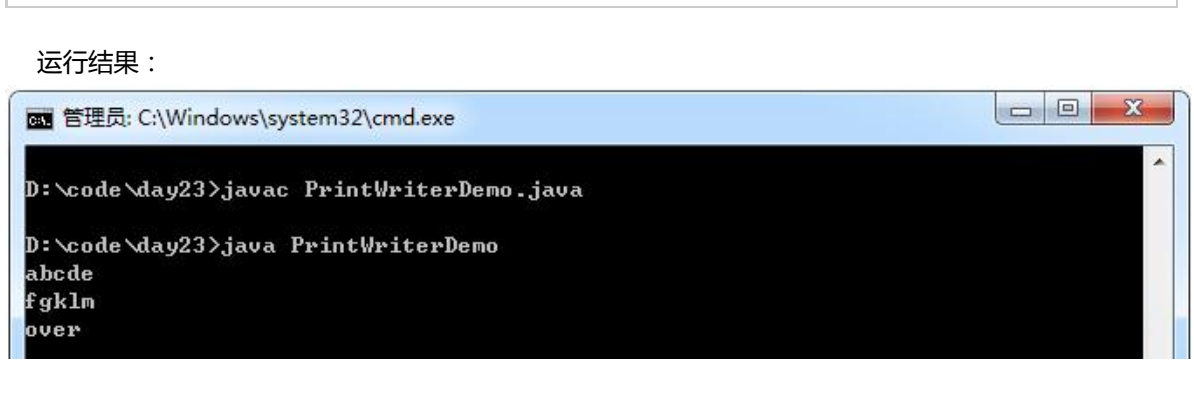
运行结果：



示例3：

```
import java.io.BufferedReader;
import java.io.FileWriter;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.io.IOException;

//写入到out.txt文件中
public class PrintWriterDemo{
       public static void main(String[] args) throws IOException {
            BufferedReader bufr = new BufferedReader(new InputStreamReader(System.in));
             //PrintWriter构造函数的第二个参数设置为true，表示自动刷新
            PrintWriter out = new PrintWriter(new FileWriter("out.txt" ),true);

            String line = null;
             while((line = bufr.readLine()) != null){
                   if("over" .equals(line))
                         break;
                  out.println(line.toUpperCase());
            }

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

运行结果：



序列流

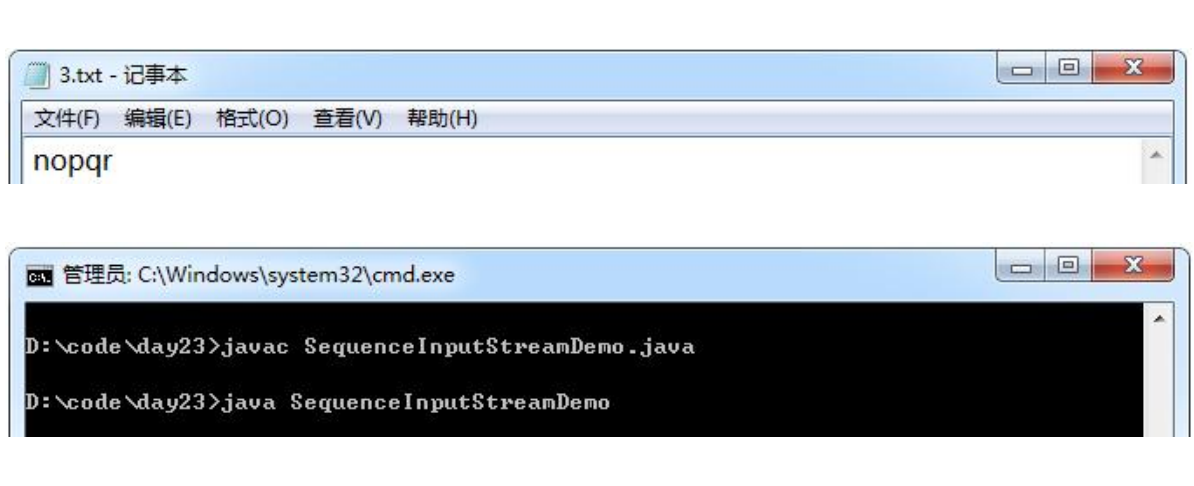
SequenceInputStream：对多个流进行合并。

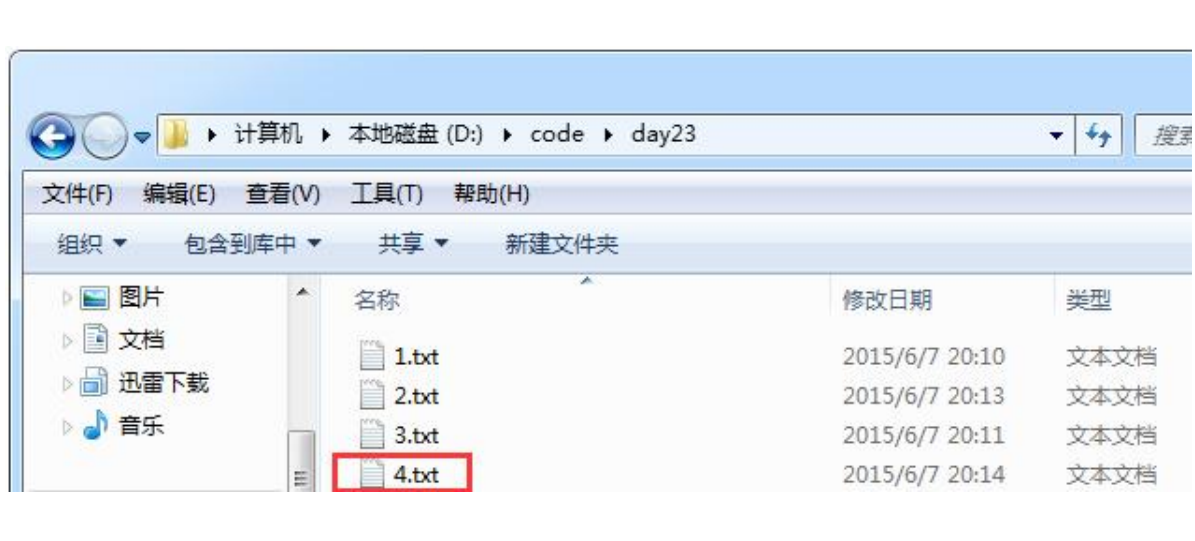
需求：将1.txt、2.txt、3、txt文件中的数据合并到一个文件中。

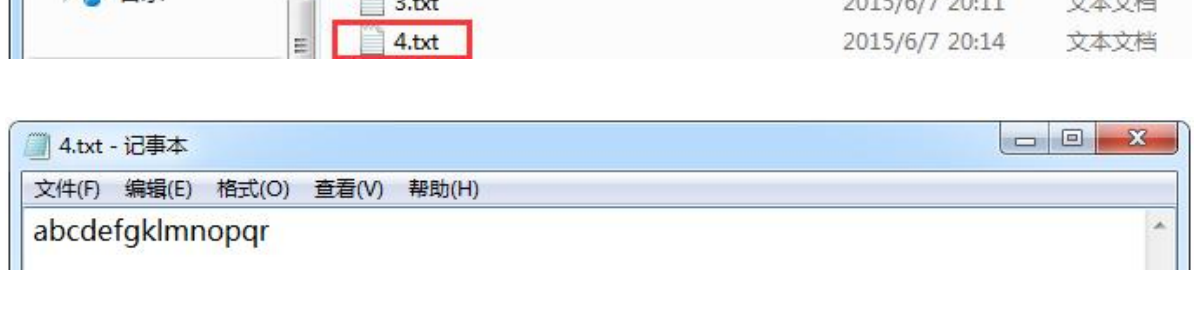
代码：

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.SequenceInputStream;
import java.util.Enumeration;
import java.util.Vector;

public class SequenceInputStreamDemo{
       public static void main(String[] args) throws Exception {
            Vector<FileInputStream> v = new Vector<FileInputStream>();

            v.add( new FileInputStream("1.txt" ));
            v.add( new FileInputStream("2.txt" ));
            v.add( new FileInputStream("3.txt" ));

            Enumeration<FileInputStream> en = v.elements();

            SequenceInputStream sis = new SequenceInputStream(en);

            FileOutputStream fos = new FileOutputStream("4.txt" );

             byte[] buf = new byte[1024];

             int len = 0;

             while((len = sis.read(buf)) != -1){
                  fos.write(buf,0,len);
            }

            fos.close();
            sis.close();
       }
}
```
复制代码

运行结果：







使用ArrayList比Vector效率更高。

改进后代码：

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.SequenceInputStream;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Enumeration;
import java.util.Iterator;

public class SequenceInputStreamDemo{
    public static void main(String[] args) throws Exception {

        ArrayList<FileInputStream> al = new ArrayList<FileInputStream>();
        for(int x = 1; x <= 3; x++){
            al.add( new FileInputStream(x + ".txt" ));
        }

        final Iterator<FileInputStream> it = al.iterator();

        Enumeration<FileInputStream> en = Collections.enumeration(al);

        /*
        //Collections工具类的enumeration方法核心代码：
        Enumeration<FileInputStream> en = new Enumeration<FileInputStream>(){
            public boolean hasMoreElements(){
                return it.hasMoreElements();
            }

            public FileInputStream nextElement(){
                return it.next();
            }
        };*/

        SequenceInputStream sis = new SequenceInputStream(en);

        FileOutputStream fos = new FileOutputStream("4.txt" );

        byte[] buf = new byte[1024];

        int len = 0;

        while((len = sis.read(buf)) != -1){
            fos.write(buf,0,len);
        }

        fos.close();
        sis.close();
    }
}
```
复制代码

~END~

~爱上海，爱黑马~