# NIO

[**http://www.cnblogs.com/Qian123/p/5256084.html**](http://www.cnblogs.com/Qian123/p/5256084.html)

Java注解了解一波~

### 缓冲区（ Buffers ）

新的 Buffer 类是常规 Java 类和通道之间的纽带。原始数据元素组成的固定长度数组，封装在

包含状态信息的对象中，存入缓冲区。缓冲区提供了一个会合点：通道既可提取放在缓冲区中的数

据（写），也可向缓冲区存入数据供读取（读）。此外，还有一种特殊类型的缓冲区，用于内存映

射文件。

### 通道（ Channels ）

NIO 新引入的最重要的抽象是通道的概念。Channel 对象模拟了通信连接，管道既可以是单向

的（进或出），也可以是双向的（进和出）。可以把通道想象成连接缓冲区和 I/O 服务的捷径。

某些情况下，软件包中的旧类可利用通道。为了能够向与文件或套接字关联的通道进行存取，

适当的地方都增加了新方法。

多数通道可工作在非块模式下，这意味着更好的可伸缩性，尤其是与选择器一同使用的时候。

### 文件锁定和内存映射文件（ File locking and memory-mapped files ）

新的 FileChannel 对象包含在 java.nio.channels 软件包内，提供许多面向文件的新特

性，其中最有趣的两个是文件锁定和内存映射文件。

在多个进程协同工作的情况下，要协调各个进程对共享数据的访问，文件锁定是必不可少的工

具。

将文件映射到内存，这样在您看来，磁盘上的文件数据就像是在内存中一样。这利用了操作系

统的虚拟内存功能，无需在内存中实际保留一份文件的拷贝，就可实现文件内容的动态高速缓存。

### 套接字（ Sockets ）

套接字通道类为使用网络套接字实现交互提供了新方法。套接字通道可工作于非块模式，并可

与选择器一同使用。因此，多个套接字可实现多路传输，管理效率也比 java.net 提供的传统套

接字更高。

三个新套接字通道，即 ServerSocketChannel、SocketChannel 和 DatagramChannel

### 选择器（ Selectors ）

选择器可实现就绪性选择。Selector 类提供了确定一或多个通道当前状态的机制。使用选择

器，借助单一线程，就可对数量庞大的活动 I/O 通道实施监控和维护。

### 正则表达式（ Regular expressions ）

新增的 java.util.regex 软件包将类似 Perl 语言的正则表达式处理机制引入 Java。这一人

们期盼已久的特性有着广泛用途。

新的正则表达式 API 之所以被看成是 NIO 的组成部分，是因 JSR 51 把它与其他 NIO 特性放在

一起作了详细说明。虽然它在许多方面与 NIO 的其他组成部分缺乏平行关系，但它在文件处理等

众多领域都是极其有用的。

### 字符集（ Character sets ）

java.nio.charsets 提供了新类用于处理字符与字节流之间的映射关系。您可以对字符转换映射方式进行选择，也可以自己创建映射。

在阅读本书的过程中，理解以下概念是非常重要的：

• 缓冲区操作

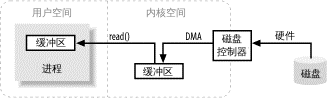
• 内核空间与用户空间

• 虚拟内存

• 分页技术

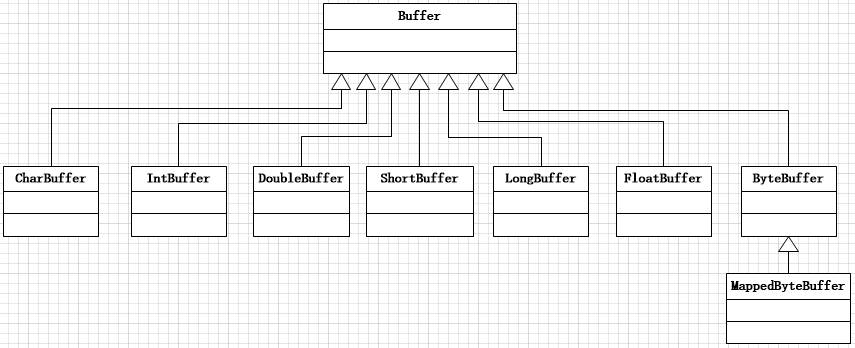
• 面向文件的 I/O 和流 I/O

• 多工 I/O（就绪性选择）

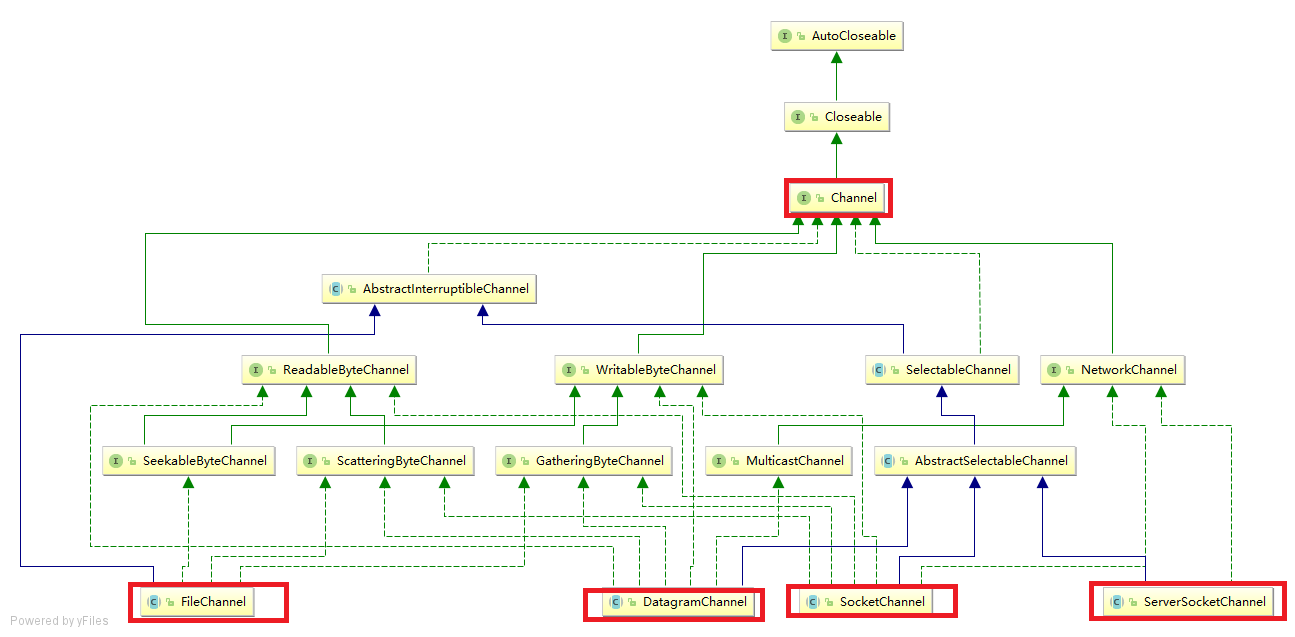


进程执行 I/O 操作，归结起来，也就是向操作系统发出请求，让它要么把缓冲区里的数据排干（写），要么用数据把缓冲区填满（读）。

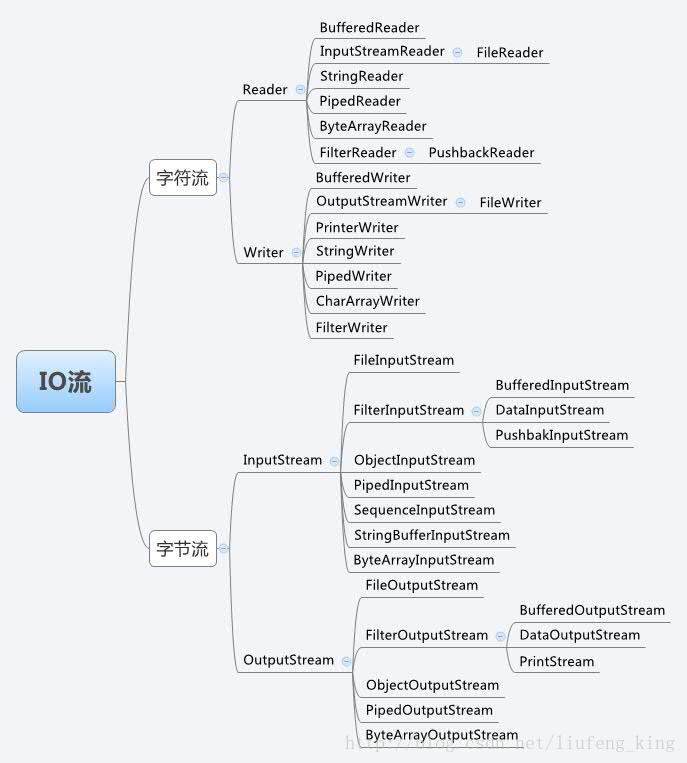
一个Buffer对象是固定数量的数据的容器。其作用是一个存储器，或者分段运输区，在这里数据可被存储并在之后用于检索。



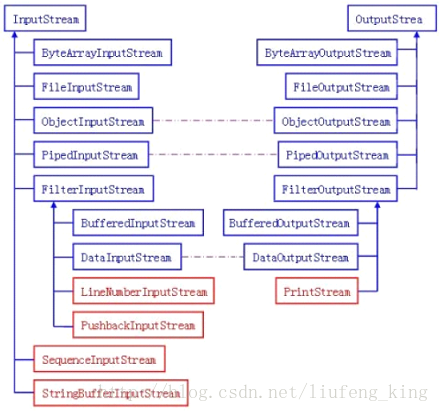
Channel 用于在字节缓冲区和位于通道另一侧的实体（通常是一个文件或套接字）之间有效地传输数据。



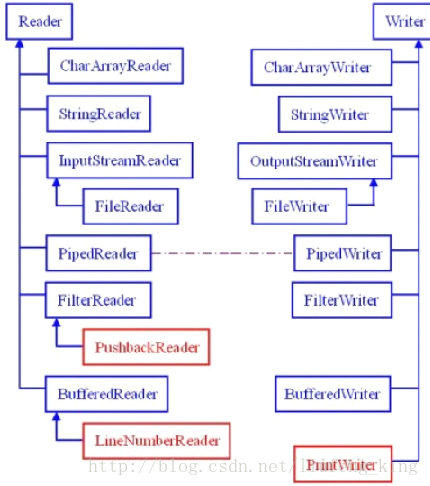
### IO流



### 字节流



### 字符流



## 从磁盘文件中读取内容

**public static void** main(**String**[] args) {  
 **String** fileName = "e:/a.txt";  
 **File** file = **new** File(fileName);  
 **InputStream** in = **null**;  
  
 **try** {  
 **System**.***out***.println("以字节为单位读取文件内容，一次读一个字节：");  
 in = **new** FileInputStream(file);// 将文件放入文件输入流中  
 **int** tempbyte;  
 **while** ((tempbyte = in.read()) != -**1**) { // 一次读一个字节,循环将内容读出来  
 **System**.***out***.write(tempbyte);  
 }  
 in.close();// 关闭文件输入流  
 } **catch** (**Exception** e) {// 捕获异常  
 e.printStackTrace();  
 **return**;  
 }  
  
 **System**.***out***.println();  
 **try** {  
 **System**.***out***.println("以字节为单位读取文件内容，一次读多个字节：");  
 // 一次读多个字节  
 **byte**[] tempbytes = **new byte**[**100**];// 声明长度为100的字节数组  
 **int** byteread = **0**;  
 in = **new** FileInputStream(fileName);// 将文件放入文件输入流中  
 // 读入多个字节到字节数组中，byteread为一次读入的字节数  
 **while** ((byteread = in.read(tempbytes)) != -**1**) {// 一次读多个字节,循环将内容读出来  
 **System**.***out***.write(tempbytes, **0**, byteread);  
 }  
 } **catch** (**Exception** e1) {// 捕获异常  
 e1.printStackTrace();  
 } **finally** {// 内容总执行  
 **if** (in != **null**) {  
 **try** {  
 in.close();// 确保文件输入流关闭  
 } **catch** (**IOException** e1) {  
 e1.printStackTrace();  
 }  
 }  
 }  
  
 **System**.***out***.println();  
 **Reader** read = **null**;  
 **try** {  
 **System**.***out***.println("以字符为单位读取文件内容，一次读一个字节：");  
 // 一次读一个字符  
 read = **new** InputStreamReader(**new** FileInputStream(file));  
 **int** tempchar;  
 **while** ((tempchar = read.read()) != -**1**) {  
 // 对于windows下，rn这两个字符在一起时，表示一个换行。但如果这两个字符分开显示时，会换两次行。因此，屏蔽掉r，或者屏蔽n。否则，将会多出很多空行。  
 **if** (((**char**) tempchar) != 'r') {// 只要是不换行就读取  
 **System**.***out***.print((**char**) tempchar);  
 }  
 }  
 read.close();  
 } **catch** (**Exception** e) {  
 e.printStackTrace();  
 }  
  
 **System**.***out***.println();  
 **try** {  
 **System**.***out***.println("以字符为单位读取文件内容，一次读多个字节：");  
 **char**[] tempchars = **new char**[**30**];  
 **int** charread = **0**;  
 read = **new** InputStreamReader(**new** FileInputStream(fileName));// 创建文件读入流  
 **while** ((charread = read.read(tempchars)) != -**1**) {// 一次读多个字符,同样屏蔽掉r不显示  
 **if** ((charread == tempchars.*length*) && (tempchars[tempchars.*length* - **1**] != 'r')) {  
 **System**.***out***.print(tempchars);  
 } **else** {  
 **for** (**int** i = **0**; i < charread; i++) {  
 **if** (tempchars[i] == 'r') {  
 **continue**;// 停止执行当前的迭代,然后退回循环开始处  
 } **else** {  
 **System**.***out***.print(tempchars[i]);  
 }  
 }  
 }  
 }  
 } **catch** (**Exception** e1) {// 捕获异常  
 e1.printStackTrace();  
 } **finally** {// 内容总执行  
 **if** (read != **null**) {  
 **try** {  
 read.close();// 确保关闭  
 } **catch** (**IOException** e1) {  
 }  
 }  
 }  
  
}

## Buffer Chanel之间互用的文件复制

|  |
| --- |
| /\*\*  \* Buffer Chanel之间互用的文件复制  \*/  **public** **class** CopyFile {  **public** **static** **void** main(String[] args) **throws** IOException {  String infile = "E:\\a.txt";  String outfile = "E:\\b.sql";  // 获取源文件和目标文件的输入输出流  @SuppressWarnings("resource")  FileInputStream fin = **new** FileInputStream(infile);  @SuppressWarnings("resource")  FileOutputStream fout = **new** FileOutputStream(outfile);  // 获取输入输出通道  FileChannel fcin = fin.getChannel();  FileChannel fcout = fout.getChannel();  // 创建缓冲区  ByteBuffer buffer = ByteBuffer.*allocate*(1024);  **while** (**true**) {  // clear方法重设缓冲区，使它可以接受读入的数据  buffer.clear();  // 从输入通道中将数据读到缓冲区  **int** r = fcin.read(buffer);  // read方法返回读取的字节数，可能为零，如果该通道已到达流的末尾，则返回-1  **if** (r == -1) {  **break**;  }  // flip方法让缓冲区可以将新读入的数据写入另一个通道  buffer.flip();  // 从输出通道中将数据写入缓冲区  fcout.write(buffer);  }  }  } |

## 读取磁盘文件readTxtUtil工具包

|  |
| --- |
| **public** **class** readTxtUtil {  **public** **static** List<String> readTxtUtil(MultipartFile file) {  List<String> list = Lists.newArrayList();  **try** {  // InputStreamReader isr = new InputStreamReader(new FileInputStream(file), "utf-8");  // 处理文件的字符集，得到文件的字符集  // cpdetector是一个识别文件的工具。常用的utf - 8 ，还有GBK，都能够很好的识别，可以到官网上下载最新的jar包，一定要将cpdetector\_1.0.8.jar，antlr-2.7.4.jar，还有chardet-1.0.jar引入其中。  CodepageDetectorProxy detector = CodepageDetectorProxy.getInstance();  detector.add(**new** ParsingDetector(**false**));  detector.add(JChardetFacade.getInstance());  detector.add(ASCIIDetector.getInstance());  detector.add(UnicodeDetector.getInstance());  Charset charset = **null**;  **try** {  // charset = detector.detectCodepage(file.toURI().toURL());  InputStream is = **new** BufferedInputStream(file.getInputStream());  charset = detector.detectCodepage(is, 8);  } **catch** (Exception e) {  e.printStackTrace();  **throw** e;  }  InputStreamReader isr = **new** InputStreamReader(file.getInputStream(), charset.name());  // InputStreamReader isr = new  // InputStreamReader(file.getInputStream(), "utf-8");  BufferedReader br = **new** BufferedReader(isr);  String lineTxt = **null**;  **while** ((lineTxt = br.readLine()) != **null**) {  // System.out.println(lineTxt);  list.add(lineTxt);  }  br.close();  } **catch** (Exception e) {  System.***out***.println("文件读取错误!");  }  **return** list;  }  } |

## JAVA NIO(五)：如何在5秒内写入10G的文本数据

<https://blog.csdn.net/yiifaa/article/details/78128363>

## JAVA NIO(六)：读取10G的文件其实很容易

<https://blog.csdn.net/yiifaa/article/details/78141622>

## java nio 常见的nio类, nio 写文件方法, nio 比io 的优势

|  |
| --- |
| **package** nio;  **import** java.io.BufferedInputStream;  **import** java.io.FileInputStream;  **import** java.io.IOException;  **import** java.io.RandomAccessFile;  **import** java.net.InetSocketAddress;  **import** java.net.ServerSocket;  **import** java.nio.ByteBuffer;  **import** java.nio.CharBuffer;  **import** java.nio.channels.DatagramChannel;  **import** java.nio.channels.FileChannel;  **import** java.nio.channels.Pipe;  **import** java.nio.channels.SelectableChannel;  **import** java.nio.channels.SelectionKey;  **import** java.nio.channels.Selector;  **import** java.nio.channels.ServerSocketChannel;  **import** java.nio.channels.SocketChannel;  **import** java.nio.charset.Charset;  **import** java.nio.charset.CharsetDecoder;  **import** java.util.Date;  **import** java.util.Iterator;  /\*\*  \*  \* NIO (2) Chanel,chanel 就是通道,ByteBuffer俗称缓冲器,ByteBuffer 底层用byte[]数组存储实现的  \* 以下函数注释中括号的数字对应如下链接中的章节,参考:http://ifeve.com/file-channel/  \*  \* NIO 和IO 的区别:  \* nio 快,io 慢.(13)unBlockReadTest()函数  \* nio 不阻塞, io阻塞, 未看出  \* nio 可以从文件的指定位置读/写, io做不到 (14)updateFile()函数  \*  \* **@author** jaloli  \*  \*/  **public** **class** FileChanelTest {  //(2,3)  **public** **static** **void** writeAndReadChanel() **throws** IOException { //write buffer,read buffer  RandomAccessFile aFile = **new** RandomAccessFile("nio-data.txt", "rw");  FileChannel inChannel = aFile.getChannel();  ByteBuffer buf = ByteBuffer.*allocate*(48);  **int** bytesRead = inChannel.read(buf);  **while** (bytesRead != -1) {  System.***out***.println("Read " + bytesRead);  buf.flip();//翻转缓冲，使其由能写变成能读  System.***out***.println("初始　position　：" +buf.position() + " limit : " + buf.limit() );  **while**(buf.hasRemaining()){  System.***out***.print( buf.position() + ":" + (**char**) buf.get() + " ");  }  buf.clear();  bytesRead = inChannel.read(buf);  System.***out***.println("\nRead " + bytesRead);  }  aFile.close();  }  //(4)分散（scatter）从Channel中读取是指在读操作时将读取的数据写入多个buffer中  //聚集（gather）写入Channel是指在写操作时将多个buffer的数据写入同一个Channel  **public** **static** **void** writeFileChanel() **throws** IOException { //write chanel  RandomAccessFile aFile = **new** RandomAccessFile("nio-data1.txt", "rw");  FileChannel inChannel = aFile.getChannel();  ByteBuffer body = ByteBuffer.*allocate*(6);  ByteBuffer header = ByteBuffer.*allocate*(120000000);  header.put(**new** **byte**[]{65,66,67,68,69,70});  body.put(**new** **byte**[]{71,72,73,74,75,76});  header.flip();  body.flip();  // System.out.println(header.getChar());  ByteBuffer[] bufferArray = {header,body};//这里就是读buffer  inChannel.write(bufferArray);  aFile.close();  }    //(12)非阻塞测试, 如151行注释, 非block 并未发生,存疑  **public** **static** **void** unBlockWriteTest() **throws** IOException { //write chanel  RandomAccessFile aFile = **new** RandomAccessFile("nio-data1.txt", "rw");  FileChannel inChannel = aFile.getChannel();  ByteBuffer body = ByteBuffer.*allocate*(6);  ByteBuffer header = ByteBuffer.*allocate*(120000000);  **byte**[] needWriteByte = **new** **byte**[10000000];  **byte**[] needWriteByte1 = **new** **byte**[10000000];  **byte**[] needWriteByte2 = **new** **byte**[10000000];  **byte**[] needWriteByte3 = **new** **byte**[10000000];  **byte**[] needWriteByte4 = **new** **byte**[10000000];  **byte**[] needWriteByte5 = **new** **byte**[10000000];  **byte**[] needWriteByte6 = **new** **byte**[10000000];  **byte**[] needWriteByte7 = **new** **byte**[10000000];  **byte**[] needWriteByte8 = **new** **byte**[10000000];  **byte**[] needWriteByte9 = **new** **byte**[10000000];  **byte**[] needWriteByte10 = **new** **byte**[10000000];  **byte**[] needWriteByte11 = **new** **byte**[10000000];  **for**(**int** i=0;i<10000000;i++){//10000000  needWriteByte[i] = 1;  needWriteByte1[i] = 1;  needWriteByte2[i] = 1;  needWriteByte3[i] = 1;  needWriteByte4[i] = 1;  needWriteByte5[i] = 1;  needWriteByte6[i] = 1;  needWriteByte7[i] = 1;  needWriteByte8[i] = 1;  needWriteByte9[i] = 1;  needWriteByte10[i] = 1;  needWriteByte11[i] = 1;  }  header.put(needWriteByte);  header.put(needWriteByte1);  header.put(needWriteByte2);  header.put(needWriteByte3);  header.put(needWriteByte4);  header.put(needWriteByte5);  header.put(needWriteByte6);  header.put(needWriteByte7);  header.put(needWriteByte8);  header.put(needWriteByte9);  header.put(needWriteByte10);  header.put(needWriteByte11);  body.put(**new** **byte**[]{71,72,73,74,75,76});//  header.flip();  body.flip();  ByteBuffer[] bufferArray = {header,body};// 这里就是读buffer  inChannel.write(bufferArray); // 这个会写117M的文件, 这个写的时候,卡住一下,写好后,再继续走下面的代码  System.***out***.println("gggggggggggggggggggg");  aFile.close();  }    //(13)nio 非阻塞读, 没看出来,但是看出来 nio 性能优于io  **public** **static** **void** unBlockReadTest() **throws** IOException { //write chanel  RandomAccessFile aFile = **new** RandomAccessFile("nio-data1.txt", "rw");  FileChannel inChannel = aFile.getChannel();  **int** \_5M = 1024\*1024\*5;  ByteBuffer dst = ByteBuffer.*allocateDirect*(1);//\_5M  Date date = **new** Date();  System.***out***.println(String.*valueOf*(date.getTime()));  **for**(**int** i=0;i<10000000;i++){  inChannel.read(dst);//这个就是一直在读文件的dst 个字节  // inChannel.position(3);  // System.out.println(inChannel.position());//以上两句有效,但是对read()方法无作用  // System.out.println(byteBufferToString(dst));  }  System.***out***.println(String.*valueOf*(date.getTime()));//以上读完耗时忽略,是因为NIO面向的是data chunks,而java.io基本上是面向byte    FileInputStream fr = **new** FileInputStream("nio-data1.txt");  BufferedInputStream bis = **new** BufferedInputStream(fr);  **byte**[] bytes = **new** **byte**[\_5M];  System.***out***.println(String.*valueOf*(date.getTime()));  **for**(**int** i=0;i<10000000;i++) {  System.***out***.println(bis.read(bytes));  }  System.***out***.println(String.*valueOf*(date.getTime()));//java io 读完耗时特别长,原因就是链接http://justjavac.iteye.com/blog/1998207 说的io 用吸管吸os给的一盆水  }    //(14)nio 可以从文件指定位置写, 从文件指定位置读, io 做不到  **public** **static** **void** updateFile() **throws** IOException {  RandomAccessFile aFile = **new** RandomAccessFile("nio-data1.txt", "rw");  FileChannel inChannel = aFile.getChannel();  ByteBuffer needInsert = ByteBuffer.*allocate*(216);  ByteBuffer readIn = ByteBuffer.*allocate*(216);  needInsert.put("好".getBytes());  needInsert.flip();//如果不加这句则无论字母汉字都是出乱码,因为后面的不可预知的字符会以utf-8等编码去解析  inChannel.write(needInsert,3);//文件指定位置插入,源文件是 gegbbbbbbbbbwqfewq, 文件插入指定位置插入字符后变为 geg好bbbbbbbbbwqfewq  inChannel.read(readIn, 3);//从指定位置读 我bbbbbbwqfewq  readIn.flip();  System.***out***.println(*byteBufferToString*(readIn));  aFile.close();  }    //(5)transferFrom / transferTo  **public** **static** **void** transferFromTest() {    }    //(6)selector 不懂干啥的， 浏览器中输入localhost:30/fdg 可以一直打印对象.代码参考：http://www.cnblogs.com/gaotianle/p/3325451.html  **public** **static** **void** selectorTest() **throws** IOException {  **int** port = 30;  ServerSocketChannel serverChannel = ServerSocketChannel.*open*();  ServerSocket serverSocket = serverChannel.socket();  Selector selector = Selector.*open*( );  serverSocket.bind (**new** InetSocketAddress(port));  serverChannel.configureBlocking(**false**);  serverChannel.register(selector, SelectionKey.***OP\_ACCEPT***);//第二个参数是四种事件。这样serverChannel 作为一个channel 就注册到selector 上了  **while** (**true**) {  **int** n = selector.select();  **if** (n == 0) {  **continue**; // nothing to do  }  Iterator it = selector.selectedKeys().iterator();  **while** (it.hasNext( )) {  SelectionKey key = (SelectionKey) it.next( );  **if** (key.isAcceptable()) {  ServerSocketChannel serverchanel = (ServerSocketChannel) key.channel();  SocketChannel channel = serverchanel.accept();  **if** (channel == **null**) {  ;//handle code, could happen  }  channel.configureBlocking (**false**);//selector.select 来选择已经注册的多个通道之一  channel.register (selector, SelectionKey.***OP\_READ***);  } **else** **if** (key.isReadable()) {  SocketChannel channel = (SocketChannel)key.channel(); //通过selector 的key 拿到channel  ByteBuffer buffer = ByteBuffer.*allocate*(48);//channel 的东西落到buffer 里,进而转化成byte，转化成string  **int** read = channel.read(buffer);  **byte**[] data = buffer.array();  String message = **new** String(data);  System.***out***.println("接收到的消息是： " + message);  } **else** **if**(key.isWritable()) {    } **else** **if**(key.isConnectable()) {  SelectableChannel chanel = key.channel();  System.***out***.println(chanel.toString());  }  it.remove( );  }  }  }  //ByteBuffer to String  **private** **static** String byteBufferToString(ByteBuffer buf) {  CharBuffer charBuffer = **null**;  **try**{  **if**(buf.position() != 0)  buf.flip(); //flip() 函数是让 ByteBuffer 对象的position 变为0. 这里不flip()打不出来ByteBuffer 缓冲区的数据  Charset charset = Charset.*forName*("UTF-8");  CharsetDecoder decoder = charset.newDecoder();  charBuffer = decoder.decode(buf);  **return** charBuffer.toString();  } **catch**(Exception ex) {  ex.printStackTrace();  }  **return** **null**;  }    //(7)FileChannel  **public** **static** **void** fileChannelTest() **throws** IOException{  //FileChannel 读数据  RandomAccessFile rFile = **new** RandomAccessFile("D:/xiecheng/EclipseWorkingspace/WorkspaceLearning/jsp\_api/src/FileChannelReadTest.txt", "rw");  FileChannel inChannel = rFile.getChannel();  ByteBuffer buf = ByteBuffer.*allocate*(48);  **int** bytesRead = inChannel.read(buf);  String readFileContent = *byteBufferToString*(buf);  System.***out***.println("读文件结果为: "+readFileContent);    //FileChannel 写数据  RandomAccessFile wFile = **new** RandomAccessFile("D:/xiecheng/EclipseWorkingspace/WorkspaceLearning/jsp\_api/src/FileChannelWriteTest.txt", "rw");  FileChannel outChannel = wFile.getChannel();  String needWrite = "我被写入文件aaa";  ByteBuffer needWriteBuf = ByteBuffer.*allocate*(48);  needWriteBuf.clear();  needWriteBuf.put(needWrite.getBytes());  needWriteBuf.flip();  **while**(needWriteBuf.hasRemaining()) {  outChannel.write(needWriteBuf);  }  System.***out***.println("写文件结果请看文件本身,写的内容是:"+needWrite);  }    //(8)SocketChannel如上面(6)    //(10)DatagramChannel实现UDP传输,使用如下两个函数, 参考http://www.cnblogs.com/bronte/articles/1966607.html  **public** **static** **void** socketChannelServerTest() **throws** IOException{  DatagramChannel channel = DatagramChannel.*open*();  channel.socket().bind(**new** InetSocketAddress(9991));  ByteBuffer buf = ByteBuffer.*allocate*(48);  buf.clear();  channel.receive(buf);  String readFileContent = *byteBufferToString*(buf);  System.***out***.println("SocketChannel 接收的结果为: "+readFileContent);  }  //这个客户端函数在另一个类中运行,方便查看客户端给服务端发的数据  **public** **static** **void** socketChannelClientTest() **throws** IOException{  DatagramChannel channel = DatagramChannel.*open*();  String newData = "New String to write to file..." + System.*currentTimeMillis*();  ByteBuffer buf = ByteBuffer.*allocate*(48);  buf.clear();  buf.put(newData.getBytes());  buf.flip();  **int** bytesSent = channel.send(buf, **new** InetSocketAddress(9991));  }  //(11)Pipe  **public** **static** **void** pipeTest() **throws** IOException {  //写  Pipe pipe = Pipe.*open*();  Pipe.SinkChannel sinkChannel = pipe.sink();  String newData = "New String to write to file..." + System.*currentTimeMillis*();  ByteBuffer needBeWriteBuf = ByteBuffer.*allocate*(48);//此时needBeWriteBuf 这个ByteBuffer 的limit 值是0  needBeWriteBuf.clear();  needBeWriteBuf.put(newData.getBytes());  System.***out***.println("needBeWriteBuf 的postion 和 limit 分别是: "+needBeWriteBuf.position() +", "+ needBeWriteBuf.limit());  needBeWriteBuf.flip(); //position 设置为0,即顶头,读写公用一个position  **while**(needBeWriteBuf.hasRemaining()) { //这个一次可能写0个到多个,所以需要重复写  **int** byteWriteNum = sinkChannel.write(needBeWriteBuf);//write 之后,needBeWriteBuf 的position 就变成了43  System.***out***.println("往sinkChannel 写的数据字节数为:" + byteWriteNum + ", 写的数据为: " + newData);  }  //读  Pipe.SourceChannel sourceChannel = pipe.source();  ByteBuffer needReadToBuf = ByteBuffer.*allocate*(48);  **int** byteReadNum = sourceChannel.read(needReadToBuf);  System.***out***.println("read 之后, postion is : "+needReadToBuf.position()); //read 一次, position 变为43  String sourceChannelContent = *byteBufferToString*(needReadToBuf);  System.***out***.println("sourceChannel 接收的结果长度为: " + byteReadNum +", sourceChannel 接收的结果为: "+sourceChannelContent);  }    //ByteBuffer 缓冲区  **public** **static** **void** byteBufferTest(){  ByteBuffer needBeWriteBuf = ByteBuffer.*allocate*(48);  needBeWriteBuf.clear();  needBeWriteBuf.put("我".getBytes());  needBeWriteBuf.put("我".getBytes());  needBeWriteBuf.put("我".getBytes());  // needBeWriteBuf.clear(); //buffer.clear() 缓冲区清空了  needBeWriteBuf.put("你".getBytes()); //正常追加到缓冲区, 我我我你    String sourceChannelContent = *byteBufferToString*(needBeWriteBuf);  System.***out***.println("ByteBuffer 缓冲区内容是 is " + sourceChannelContent);  }    **public** **static** **void** main(String[] args) **throws** IOException {  // writeAndReadChanel();  // writeFileChanel();  // selectorTest();  // fileChannelTest();  // socketChannelServerTest();  // pipeTest();  // byteBufferTest();  // unBlockWriteTest();  // unBlockReadTest();  *updateFile*();  }  } |

# <https://blog.csdn.net/tianzhaoai/article/details/50709035>

# java中的IO整理

### 1:创建一个新文件

1. **import** java.io.\*;
2. **class** hello{
3. **public** **static** **void** main(String[] args) {
4. File f=**new** File("D:\\hello.txt");
5. **try**{
6. f.createNewFile();
7. }**catch** (Exception e) {
8. e.printStackTrace();
9. }
10. }
11. }

### 2:File类的两个常量

1. **import** java.io.\*;
2. **class** hello{
3. **public** **static** **void** main(String[] args) {
4. System.out.println(File.separator);
5. System.out.println(File.pathSeparator);
6. }
7. }

\

;

1. **import** java.io.\*;
2. **class** hello{
3. **public** **static** **void** main(String[] args) {
4. String fileName="D:"+File.separator+"hello.txt";
5. File f=**new** File(fileName);
6. **try**{
7. f.createNewFile();
8. }**catch** (Exception e) {
9. e.printStackTrace();
10. }
11. }
12. }

### ****3:删除一个文件****

1. /\*\*
2. \* 删除一个文件
3. \* \*/
4. **import** java.io.\*;
5. **class** hello{
6. **public** **static** **void** main(String[] args) {
7. String fileName="D:"+File.separator+"hello.txt";
8. File f=**new** File(fileName);
9. **if**(f.exists()){
10. f.delete();
11. }**else**{
12. System.out.println("文件不存在");
13. }
15. }
16. }

### ****4:创建一个文件夹****

1. /\*\*
2. \* 创建一个文件夹
3. \* \*/
4. **import** java.io.\*;
5. **class** hello{
6. **public** **static** **void** main(String[] args) {
7. String fileName="D:"+File.separator+"hello";
8. File f=**new** File(fileName);
9. f.mkdir();
10. }
11. }

### ****5:列出指定目录的全部文件（包括隐藏文件）：****

1. /\*\*
2. \* 使用list列出指定目录的全部文件
3. \* \*/
4. **import** java.io.\*;
5. **class** hello{
6. **public** **static** **void** main(String[] args) {
7. String fileName="D:"+File.separator;
8. File f=**new** File(fileName);
9. String[] str=f.list();
10. **for** (**int** i = 0; i < str.length; i++) {
11. System.out.println(str[i]);
12. }
13. }
14. }

### ****6:列出指定目录的全部文件（包括隐藏文件）：****

1. /\*\*
2. \* 使用listFiles列出指定目录的全部文件
3. \* listFiles输出的是完整路径
4. \* \*/
5. **import** java.io.\*;
6. **class** hello{
7. **public** **static** **void** main(String[] args) {
8. String fileName="D:"+File.separator;
9. File f=**new** File(fileName);
10. File[] str=f.listFiles();
11. **for** (**int** i = 0; i < str.length; i++) {
12. System.out.println(str[i]);
13. }
14. }
15. }

### ****7:判断一个指定的路径是否为目录****

1. /\*\*
2. \* 使用isDirectory判断一个指定的路径是否为目录
3. \* \*/
4. **import** java.io.\*;
5. **class** hello{
6. **public** **static** **void** main(String[] args) {
7. String fileName="D:"+File.separator;
8. File f=**new** File(fileName);
9. **if**(f.isDirectory()){
10. System.out.println("YES");
11. }**else**{
12. System.out.println("NO");
13. }
14. }
15. }

### ****8:搜索指定目录的全部内容****

1. /\*\*
2. \* 列出指定目录的全部内容
3. \* \*/
4. **import** java.io.\*;
5. **class** hello{
6. **public** **static** **void** main(String[] args) {
7. String fileName="D:"+File.separator;
8. File f=**new** File(fileName);
9. print(f);
10. }
11. **public** **static** **void** print(File f){
12. **if**(f!=**null**){
13. **if**(f.isDirectory()){
14. File[] fileArray=f.listFiles();
15. **if**(fileArray!=**null**){
16. **for** (**int** i = 0; i < fileArray.length; i++) {
17. //递归调用
18. print(fileArray[i]);
19. }
20. }
21. }
22. **else**{
23. System.out.println(f);
24. }
25. }
26. }
27. }

### ****9:使用RandomAccessFile写入文件****

1. /\*\*
2. \* 使用RandomAccessFile写入文件
3. \* \*/
4. **import** java.io.\*;
5. **class** hello{
6. **public** **static** **void** main(String[] args) **throws** IOException {
7. String fileName="D:"+File.separator+"hello.txt";
8. File f=**new** File(fileName);
9. RandomAccessFile demo=**new** RandomAccessFile(f,"rw");
10. demo.writeBytes("asdsad");
11. demo.writeInt(12);
12. demo.writeBoolean(**true**);
13. demo.writeChar('A');
14. demo.writeFloat(1.21f);
15. demo.writeDouble(12.123);
16. demo.close();
17. }
18. }

## 字节流

### 1:向文件中写入字符串

1. /\*\*
2. \* 字节流
3. \* 向文件中写入字符串
4. \* \*/
5. **import** java.io.\*;
6. **class** hello{
7. **public** **static** **void** main(String[] args) **throws** IOException {
8. String fileName="D:"+File.separator+"hello.txt";
9. File f=**new** File(fileName);
10. OutputStream out =**new** FileOutputStream(f);
11. String str="你好";
12. **byte**[] b=str.getBytes();
13. out.write(b);
14. out.close();
15. }
16. }

### 2:当然也可以一个字节一个字节的写。

1. /\*\*
2. \* 字节流
3. \* 向文件中一个字节一个字节的写入字符串
4. \* \*/
5. **import** java.io.\*;
6. **class** hello{
7. **public** **static** **void** main(String[] args) **throws** IOException {
8. String fileName="D:"+File.separator+"hello.txt";
9. File f=**new** File(fileName);
10. OutputStream out =**new** FileOutputStream(f);
11. String str="你好";
12. **byte**[] b=str.getBytes();
13. **for** (**int** i = 0; i < b.length; i++) {
14. out.write(b[i]);
15. }
16. out.close();
17. }
18. }

### 3:向文件中追加新内容：

1. /\*\*
2. \* 字节流
3. \* 向文件中追加新内容：
4. \* \*/
5. **import** java.io.\*;
6. **class** hello{
7. **public** **static** **void** main(String[] args) **throws** IOException {
8. String fileName="D:"+File.separator+"hello.txt";
9. File f=**new** File(fileName);
10. OutputStream out =**new** FileOutputStream(f,**true**);
11. String str="Rollen";
12. //String str="\r\nRollen";  可以换行
13. **byte**[] b=str.getBytes();
14. **for** (**int** i = 0; i < b.length; i++) {
15. out.write(b[i]);
16. }
17. out.close();
18. }
19. }

### 4:读取文件内容

1. /\*\*
2. \* 字节流
3. \* 读文件内容
4. \* \*/
5. **import** java.io.\*;
6. **class** hello{
7. **public** **static** **void** main(String[] args) **throws** IOException {
8. String fileName="D:"+File.separator+"hello.txt";
9. File f=**new** File(fileName);
10. InputStream in=**new** FileInputStream(f);
11. **byte**[] b=**new** **byte**[1024];
12. in.read(b);
13. in.close();
14. System.out.println(**new** String(b));
15. }
16. }

### 5:但是这个例子读取出来会有大量的空格，我们可以利用in.read(b);的返回值来设计程序。如下：

1. /\*\*
2. \* 字节流
3. \* 读文件内容
4. \* \*/
5. **import** java.io.\*;
6. **class** hello{
7. **public** **static** **void** main(String[] args) **throws** IOException {
8. String fileName="D:"+File.separator+"hello.txt";
9. File f=**new** File(fileName);
10. InputStream in=**new** FileInputStream(f);
11. **byte**[] b=**new** **byte**[1024];
12. **int** len=in.read(b);
13. in.close();
14. System.out.println("读入长度为："+len);
15. System.out.println(**new** String(b,0,len));
16. }
17. }

### 6:将上面的例子改为一个一个读：

1. /\*\*
2. \* 字节流
3. \* 读文件内容,节省空间
4. \* \*/
5. **import** java.io.\*;
6. **class** hello{
7. **public** **static** **void** main(String[] args) **throws** IOException {
8. String fileName="D:"+File.separator+"hello.txt";
9. File f=**new** File(fileName);
10. InputStream in=**new** FileInputStream(f);
11. **byte**[] b=**new** **byte**[(**int**)f.length()];
12. **for** (**int** i = 0; i < b.length; i++) {
13. b[i]=(**byte**)in.read();
14. }
15. in.close();
16. System.out.println(**new** String(b));
17. }
18. }

### 7:细心的读者可能会发现，上面的几个例子都是在知道文件的内容多大，然后才展开的，有时候我们不知道文件有多大，这种情况下，我们需要判断是否独到文件的末尾。

1. /\*\*
2. \* 字节流
3. \*读文件
4. \* \*/
5. **import** java.io.\*;
6. **class** hello{
7. **public** **static** **void** main(String[] args) **throws** IOException {
8. String fileName="D:"+File.separator+"hello.txt";
9. File f=**new** File(fileName);
10. InputStream in=**new** FileInputStream(f);
11. **byte**[] b=**new** **byte**[1024];
12. **int** count =0;
13. **int** temp=0;
14. **while**((temp=in.read())!=(-1)){
15. b[count++]=(**byte**)temp;
16. }
17. in.close();
18. System.out.println(**new** String(b));
19. }
20. }

## 字符流

### 1:向文件中写入数据

1. /\*\*
2. \* 字符流
3. \* 写入数据
4. \* \*/
5. **import** java.io.\*;
6. **class** hello{
7. **public** **static** **void** main(String[] args) **throws** IOException {
8. String fileName="D:"+File.separator+"hello.txt";
9. File f=**new** File(fileName);
10. Writer out =**new** FileWriter(f);
11. String str="hello";
12. out.write(str);
13. out.close();
14. }
15. }

### 2:从文件中读内容：

1. /\*\*
2. \* 字符流
3. \* 从文件中读出内容
4. \* \*/
5. **import** java.io.\*;
6. **class** hello{
7. **public** **static** **void** main(String[] args) **throws** IOException {
8. String fileName="D:"+File.separator+"hello.txt";
9. File f=**new** File(fileName);
10. **char**[] ch=**new** **char**[100];
11. Reader read=**new** FileReader(f);
12. **int** count=read.read(ch);
13. read.close();
14. System.out.println("读入的长度为："+count);
15. System.out.println("内容为"+**new** String(ch,0,count));
16. }
17. }

### 3:当然最好采用循环读取的方式，因为我们有时候不知道文件到底有多大。

1. /\*\*
2. \* 字符流
3. \* 从文件中读出内容
4. \* \*/
5. **import** java.io.\*;
6. **class** hello{
7. **public** **static** **void** main(String[] args) **throws** IOException {
8. String fileName="D:"+File.separator+"hello.txt";
9. File f=**new** File(fileName);
10. **char**[] ch=**new** **char**[100];
11. Reader read=**new** FileReader(f);
12. **int** temp=0;
13. **int** count=0;
14. **while**((temp=read.read())!=(-1)){
15. ch[count++]=(**char**)temp;
16. }
17. read.close();
18. System.out.println("内容为"+**new** String(ch,0,count));
19. }
20. }

### 4:下面我们使用程序来复制文件吧。

基本思路还是从一个文件中读入内容，边读边写入另一个文件，就是这么简单。、

1. /\*\*
2. \* 文件的复制
3. \* \*/
4. **import** java.io.\*;
5. **class** hello{
6. **public** **static** **void** main(String[] args) **throws** IOException {
7. **if**(args.length!=2){
8. System.out.println("命令行参数输入有误，请检查");
9. System.exit(1);
10. }
11. File file1=**new** File(args[0]);
12. File file2=**new** File(args[1]);
14. **if**(!file1.exists()){
15. System.out.println("被复制的文件不存在");
16. System.exit(1);
17. }
18. InputStream input=**new** FileInputStream(file1);
19. OutputStream output=**new** FileOutputStream(file2);
20. **if**((input!=**null**)&&(output!=**null**)){
21. **int** temp=0;
22. **while**((temp=input.read())!=(-1)){
23. output.write(temp);
24. }
25. }
26. input.close();
27. output.close();
28. }
29. }

### 5:将字节输出流转化为字符输出流

1. /\*\*
2. \* 将字节输出流转化为字符输出流
3. \* \*/
4. **import** java.io.\*;
5. **class** hello{
6. **public** **static** **void** main(String[] args) **throws** IOException {
7. String fileName= "d:"+File.separator+"hello.txt";
8. File file=**new** File(fileName);
9. Writer out=**new** OutputStreamWriter(**new** FileOutputStream(file));
10. out.write("hello");
11. out.close();
12. }
13. }

### 6:将字节输入流变为字符输入流

1. /\*\*
2. \* 将字节输入流变为字符输入流
3. \* \*/
4. **import** java.io.\*;
5. **class** hello{
6. **public** **static** **void** main(String[] args) **throws** IOException {
7. String fileName= "d:"+File.separator+"hello.txt";
8. File file=**new** File(fileName);
9. Reader read=**new** InputStreamReader(**new** FileInputStream(file));
10. **char**[] b=**new** **char**[100];
11. **int** len=read.read(b);
12. System.out.println(**new** String(b,0,len));
13. read.close();
14. }
15. }

### 7:使用内存操作流将一个大写字母转化为小写字母

1. /\*\*
2. \* 使用内存操作流将一个大写字母转化为小写字母
3. \* \*/
4. **import** java.io.\*;
5. **class** hello{
6. **public** **static** **void** main(String[] args) **throws** IOException {
7. String str="ROLLENHOLT";
8. ByteArrayInputStream input=**new** ByteArrayInputStream(str.getBytes());
9. ByteArrayOutputStream output=**new** ByteArrayOutputStream();
10. **int** temp=0;
11. **while**((temp=input.read())!=-1){
12. **char** ch=(**char**)temp;
13. output.write(Character.toLowerCase(ch));
14. }
15. String outStr=output.toString();
16. input.close();
17. output.close();
18. System.out.println(outStr);
19. }
20. }

## 管道流

管道流主要可以进行两个线程之间的通信。

PipedOutputStream 管道输出流

PipedInputStream 管道输入流

### 1:验证管道流

1. /\*\*
2. \* 验证管道流
3. \* \*/
4. **import** java.io.\*;
6. /\*\*
7. \* 消息发送类
8. \* \*/
9. **class** Send **implements** Runnable{
10. **private** PipedOutputStream out=**null**;
11. **public** Send() {
12. out=**new** PipedOutputStream();
13. }
14. **public** PipedOutputStream getOut(){
15. **return** **this**.out;
16. }
17. **public** **void** run(){
18. String message="hello , Rollen";
19. **try**{
20. out.write(message.getBytes());
21. }**catch** (Exception e) {
22. e.printStackTrace();
23. }**try**{
24. out.close();
25. }**catch** (Exception e) {
26. e.printStackTrace();
27. }
28. }
29. }
31. /\*\*
32. \* 接受消息类
33. \* \*/
34. **class** Recive **implements** Runnable{
35. **private** PipedInputStream input=**null**;
36. **public** Recive(){
37. **this**.input=**new** PipedInputStream();
38. }
39. **public** PipedInputStream getInput(){
40. **return** **this**.input;
41. }
42. **public** **void** run(){
43. **byte**[] b=**new** **byte**[1000];
44. **int** len=0;
45. **try**{
46. len=**this**.input.read(b);
47. }**catch** (Exception e) {
48. e.printStackTrace();
49. }**try**{
50. input.close();
51. }**catch** (Exception e) {
52. e.printStackTrace();
53. }
54. System.out.println("接受的内容为 "+(**new** String(b,0,len)));
55. }
56. }
57. /\*\*
58. \* 测试类
59. \* \*/
60. **class** hello{
61. **public** **static** **void** main(String[] args) **throws** IOException {
62. Send send=**new** Send();
63. Recive recive=**new** Recive();
64. **try**{
65. //管道连接
66. send.getOut().connect(recive.getInput());
67. }**catch** (Exception e) {
68. e.printStackTrace();
69. }
70. **new** Thread(send).start();
71. **new** Thread(recive).start();
72. }
73. }

## 打印流

### 1:

1. /\*\*
2. \* 使用PrintStream进行输出
3. \* \*/
4. **import** java.io.\*;
6. **class** hello {
7. **public** **static** **void** main(String[] args) **throws** IOException {
8. PrintStream print = **new** PrintStream(**new** FileOutputStream(**new** File("d:"
9. + File.separator + "hello.txt")));
10. print.println(**true**);
11. print.println("Rollen");
12. print.close();
13. }
14. }

### 2:当然也可以格式化输出

1. /\*\*
2. \* 使用PrintStream进行输出
3. \* 并进行格式化
4. \* \*/
5. **import** java.io.\*;
6. **class** hello {
7. **public** **static** **void** main(String[] args) **throws** IOException {
8. PrintStream print = **new** PrintStream(**new** FileOutputStream(**new** File("d:"
9. + File.separator + "hello.txt")));
10. String name="Rollen";
11. **int** age=20;
12. print.printf("姓名：%s. 年龄：%d.",name,age);
13. print.close();
14. }
15. }

### 3:使用OutputStream向屏幕上输出内容

1. /\*\*
2. \* 使用OutputStream向屏幕上输出内容
3. \* \*/
4. **import** java.io.\*;
5. **class** hello {
6. **public** **static** **void** main(String[] args) **throws** IOException {
7. OutputStream out=System.out;
8. **try**{
9. out.write("hello".getBytes());
10. }**catch** (Exception e) {
11. e.printStackTrace();
12. }
13. **try**{
14. out.close();
15. }**catch** (Exception e) {
16. e.printStackTrace();
17. }
18. }
19. }

### 4:输入输出重定向

1. **import** java.io.File;
2. **import** java.io.FileNotFoundException;
3. **import** java.io.FileOutputStream;
4. **import** java.io.PrintStream;
6. /\*\*
7. \* 为System.out.println()重定向输出
8. \* \*/
9. **public** **class** systemDemo{
10. **public** **static** **void** main(String[] args){
11. // 此刻直接输出到屏幕
12. System.out.println("hello");
13. File file = **new** File("d:" + File.separator + "hello.txt");
14. **try**{
15. System.setOut(**new** PrintStream(**new** FileOutputStream(file)));
16. }**catch**(FileNotFoundException e){
17. e.printStackTrace();
18. }
19. System.out.println("这些内容在文件中才能看到哦！");
20. }
21. }

eclipse的控制台输出的是hello。然后当我们查看d盘下面的hello.txt文件的时候，会在里面看到：这些内容在文件中才能看到哦！

1. **import** java.io.File;
2. **import** java.io.FileNotFoundException;
3. **import** java.io.FileOutputStream;
4. **import** java.io.PrintStream;
6. /\*\*
7. \* System.err重定向 这个例子也提示我们可以使用这种方法保存错误信息
8. \* \*/
9. **public** **class** systemErr{
10. **public** **static** **void** main(String[] args){
11. File file = **new** File("d:" + File.separator + "hello.txt");
12. System.err.println("这些在控制台输出");
13. **try**{
14. System.setErr(**new** PrintStream(**new** FileOutputStream(file)));
15. }**catch**(FileNotFoundException e){
16. e.printStackTrace();
17. }
18. System.err.println("这些在文件中才能看到哦！");
19. }
20. }

你会在eclipse的控制台看到红色的输出：“这些在控制台输出”，然后在d盘下面的hello.txt中会看到：这些在文件中才能看到哦！

1. **import** java.io.File;
2. **import** java.io.FileInputStream;
3. **import** java.io.FileNotFoundException;
4. **import** java.io.IOException;
6. /\*\*
7. \* System.in重定向
8. \* \*/
9. **public** **class** systemIn{
10. **public** **static** **void** main(String[] args){
11. File file = **new** File("d:" + File.separator + "hello.txt");
12. **if**(!file.exists()){
13. **return**;
14. }**else**{
15. **try**{
16. System.setIn(**new** FileInputStream(file));
17. }**catch**(FileNotFoundException e){
18. e.printStackTrace();
19. }
20. **byte**[] bytes = **new** **byte**[1024];
21. **int** len = 0;
22. **try**{
23. len = System.in.read(bytes);
24. }**catch**(IOException e){
25. e.printStackTrace();
26. }
27. System.out.println("读入的内容为：" + **new** String(bytes, 0, len));
28. }
29. }
30. }

前提是我的d盘下面的hello.txt中的内容是：“这些文件中的内容哦！”，然后运行程序，输出的结果为：读入的内容为：这些文件中的内容哦！

### 5:BufferedReader的小例子

1. BufferedReader buf = **new** BufferedReader(**new** InputStreamReader(System.in));
2. **import** java.io.BufferedReader;
3. **import** java.io.IOException;
4. **import** java.io.InputStreamReader;
6. /\*\*
7. \* 使用缓冲区从键盘上读入内容
8. \* \*/
9. **public** **class** BufferedReaderDemo{
10. **public** **static** **void** main(String[] args){
11. BufferedReader buf = **new** BufferedReader(
12. **new** InputStreamReader(System.in));
13. String str = **null**;
14. System.out.println("请输入内容");
15. **try**{
16. str = buf.readLine();
17. }**catch**(IOException e){
18. e.printStackTrace();
19. }
20. System.out.println("你输入的内容是：" + str);
21. }
22. }

### 6:Scanner类

其实我们比较常用的是采用Scanner类来进行数据输入，下面来给一个Scanner的例子吧

1. **import** java.util.Scanner;
3. /\*\*
4. \* Scanner的小例子，从键盘读数据
5. \* \*/
6. **public** **class** ScannerDemo{
7. **public** **static** **void** main(String[] args){
8. Scanner sca = **new** Scanner(System.in);
9. // 读一个整数
10. **int** temp = sca.nextInt();
11. System.out.println(temp);
12. //读取浮点数
13. **float** flo=sca.nextFloat();
14. System.out.println(flo);
15. //读取字符
16. //...等等的，都是一些太基础的，就不师范了。
17. }
18. }

### 7:其实Scanner可以接受任何的输入流

下面给一个使用Scanner类从文件中读出内容

1. **import** java.io.File;
2. **import** java.io.FileNotFoundException;
3. **import** java.util.Scanner;
5. /\*\*
6. \* Scanner的小例子，从文件中读内容
7. \* \*/
8. **public** **class** ScannerDemo{
9. **public** **static** **void** main(String[] args){
11. File file = **new** File("d:" + File.separator + "hello.txt");
12. Scanner sca = **null**;
13. **try**{
14. sca = **new** Scanner(file);
15. }**catch**(FileNotFoundException e){
16. e.printStackTrace();
17. }
18. String str = sca.next();
19. System.out.println("从文件中读取的内容是：" + str);
20. }
21. }

## 数据操作流DataOutputStream、DataInputStream类

### 1:

1. **import** java.io.DataOutputStream;
2. **import** java.io.File;
3. **import** java.io.FileOutputStream;
4. **import** java.io.IOException;
6. **public** **class** DataOutputStreamDemo{
7. **public** **static** **void** main(String[] args) **throws** IOException{
8. File file = **new** File("d:" + File.separator + "hello.txt");
9. **char**[] ch = { 'A', 'B', 'C' };
10. DataOutputStream out = **null**;
11. out = **new** DataOutputStream(**new** FileOutputStream(file));
12. **for**(**char** temp : ch){
13. out.writeChar(temp);
14. }
15. out.close();
16. }
17. }

### 2:使用DataInputStream读出内容

1. **import** java.io.DataInputStream;
2. **import** java.io.File;
3. **import** java.io.FileInputStream;
4. **import** java.io.IOException;
6. **public** **class** DataOutputStreamDemo{
7. **public** **static** **void** main(String[] args) **throws** IOException{
8. File file = **new** File("d:" + File.separator + "hello.txt");
9. DataInputStream input = **new** DataInputStream(**new** FileInputStream(file));
10. **char**[] ch = **new** **char**[10];
11. **int** count = 0;
12. **char** temp;
13. **while**((temp = input.readChar()) != 'C'){
14. ch[count++] = temp;
15. }
16. System.out.println(ch);
17. }
18. }

## 合并流 SequenceInputStream

SequenceInputStream主要用来将2个流合并在一起，比如将两个txt中的内容合并为另外一个txt。下面给出一个实例：

1. **import** java.io.File;
2. **import** java.io.FileInputStream;
3. **import** java.io.FileOutputStream;
4. **import** java.io.IOException;
5. **import** java.io.InputStream;
6. **import** java.io.OutputStream;
7. **import** java.io.SequenceInputStream;
9. /\*\*
10. \* 将两个文本文件合并为另外一个文本文件
11. \* \*/
12. **public** **class** SequenceInputStreamDemo{
13. **public** **static** **void** main(String[] args) **throws** IOException{
14. File file1 = **new** File("d:" + File.separator + "hello1.txt");
15. File file2 = **new** File("d:" + File.separator + "hello2.txt");
16. File file3 = **new** File("d:" + File.separator + "hello.txt");
17. InputStream input1 = **new** FileInputStream(file1);
18. InputStream input2 = **new** FileInputStream(file2);
19. OutputStream output = **new** FileOutputStream(file3);
20. // 合并流
21. SequenceInputStream sis = **new** SequenceInputStream(input1, input2);
22. **int** temp = 0;
23. **while**((temp = sis.read()) != -1){
24. output.write(temp);
25. }
26. input1.close();
27. input2.close();
28. output.close();
29. sis.close();
30. }
31. }

#### 文件压缩 ZipOutputStream类

先举一个压缩单个文件的例子吧：

1. **import** java.io.File;
2. **import** java.io.FileInputStream;
3. **import** java.io.FileOutputStream;
4. **import** java.io.IOException;
5. **import** java.io.InputStream;
6. **import** java.util.zip.ZipEntry;
7. **import** java.util.zip.ZipOutputStream;
9. **public** **class** ZipOutputStreamDemo1{
10. **public** **static** **void** main(String[] args) **throws** IOException{
11. File file = **new** File("d:" + File.separator + "hello.txt");
12. File zipFile = **new** File("d:" + File.separator + "hello.zip");
13. InputStream input = **new** FileInputStream(file);
14. ZipOutputStream zipOut = **new** ZipOutputStream(**new** FileOutputStream(
15. zipFile));
16. zipOut.putNextEntry(**new** ZipEntry(file.getName()));
17. // 设置注释
18. zipOut.setComment("hello");
19. **int** temp = 0;
20. **while**((temp = input.read()) != -1){
21. zipOut.write(temp);
22. }
23. input.close();
24. zipOut.close();
25. }
26. }

上面的这个例子测试的是压缩单个文件，下面的们来看看如何压缩多个文件。

1. **import** java.io.File;
2. **import** java.io.FileInputStream;
3. **import** java.io.FileOutputStream;
4. **import** java.io.IOException;
5. **import** java.io.InputStream;
6. **import** java.util.zip.ZipEntry;
7. **import** java.util.zip.ZipOutputStream;
9. /\*\*
10. \* 一次性压缩多个文件
11. \* \*/
12. **public** **class** ZipOutputStreamDemo2{
13. **public** **static** **void** main(String[] args) **throws** IOException{
14. // 要被压缩的文件夹
15. File file = **new** File("d:" + File.separator + "temp");
16. File zipFile = **new** File("d:" + File.separator + "zipFile.zip");
17. InputStream input = **null**;
18. ZipOutputStream zipOut = **new** ZipOutputStream(**new** FileOutputStream(
19. zipFile));
20. zipOut.setComment("hello");
21. **if**(file.isDirectory()){
22. File[] files = file.listFiles();
23. **for**(**int** i = 0; i < files.length; ++i){
24. input = **new** FileInputStream(files[i]);
25. zipOut.putNextEntry(**new** ZipEntry(file.getName()
26. + File.separator + files[i].getName()));
27. **int** temp = 0;
28. **while**((temp = input.read()) != -1){
29. zipOut.write(temp);
30. }
31. input.close();
32. }
33. }
34. zipOut.close();
35. }
36. }

大家自然想到，既然能压缩，自然能解压缩，在谈解压缩之前，我们会用到一个ZipFile类，先给一个这个例子吧。java中的每一个压缩文件都是可以使用ZipFile来进行表示的

1. **import** java.io.File;
2. **import** java.io.IOException;
3. **import** java.util.zip.ZipFile;
5. /\*\*
6. \* ZipFile演示
7. \* \*/
8. **public** **class** ZipFileDemo{
9. **public** **static** **void** main(String[] args) **throws** IOException{
10. File file = **new** File("d:" + File.separator + "hello.zip");
11. ZipFile zipFile = **new** ZipFile(file);
12. System.out.println("压缩文件的名称为：" + zipFile.getName());
13. }
14. }

现在我们呢是时候来看看如何加压缩文件了，和之前一样，先让我们来解压单个压缩文件（也就是压缩文件中只有一个文件的情况），我们采用前面的例子产生的压缩文件hello.zip

1. **import** java.io.File;
2. **import** java.io.FileOutputStream;
3. **import** java.io.IOException;
4. **import** java.io.InputStream;
5. **import** java.io.OutputStream;
6. **import** java.util.zip.ZipEntry;
7. **import** java.util.zip.ZipFile;
9. /\*\*
10. \* 解压缩文件（压缩文件中只有一个文件的情况）
11. \* \*/
12. **public** **class** ZipFileDemo2{
13. **public** **static** **void** main(String[] args) **throws** IOException{
14. File file = **new** File("d:" + File.separator + "hello.zip");
15. File outFile = **new** File("d:" + File.separator + "unZipFile.txt");
16. ZipFile zipFile = **new** ZipFile(file);
17. ZipEntry entry = zipFile.getEntry("hello.txt");
18. InputStream input = zipFile.getInputStream(entry);
19. OutputStream output = **new** FileOutputStream(outFile);
20. **int** temp = 0;
21. **while**((temp = input.read()) != -1){
22. output.write(temp);
23. }
24. input.close();
25. output.close();
26. }
27. }

## ZipInputStream类

当我们需要解压缩多个文件的时候，ZipEntry就无法使用了，如果想操作更加复杂的压缩文件，我们就必须使用ZipInputStream类

1. **import** java.io.File;
2. **import** java.io.FileInputStream;
3. **import** java.io.FileOutputStream;
4. **import** java.io.IOException;
5. **import** java.io.InputStream;
6. **import** java.io.OutputStream;
7. **import** java.util.zip.ZipEntry;
8. **import** java.util.zip.ZipFile;
9. **import** java.util.zip.ZipInputStream;
11. /\*\*
12. \* 解压缩一个压缩文件中包含多个文件的情况
13. \* \*/
14. **public** **class** ZipFileDemo3{
15. **public** **static** **void** main(String[] args) **throws** IOException{
16. File file = **new** File("d:" + File.separator + "zipFile.zip");
17. File outFile = **null**;
18. ZipFile zipFile = **new** ZipFile(file);
19. ZipInputStream zipInput = **new** ZipInputStream(**new** FileInputStream(file));
20. ZipEntry entry = **null**;
21. InputStream input = **null**;
22. OutputStream output = **null**;
23. **while**((entry = zipInput.getNextEntry()) != **null**){
24. System.out.println("解压缩" + entry.getName() + "文件");
25. outFile = **new** File("d:" + File.separator + entry.getName());
26. **if**(!outFile.getParentFile().exists()){
27. outFile.getParentFile().mkdir();
28. }
29. **if**(!outFile.exists()){
30. outFile.createNewFile();
31. }
32. input = zipFile.getInputStream(entry);
33. output = **new** FileOutputStream(outFile);
34. **int** temp = 0;
35. **while**((temp = input.read()) != -1){
36. output.write(temp);
37. }
38. input.close();
39. output.close();
40. }
41. }
42. }

## PushBackInputStream回退流

1. **import** java.io.ByteArrayInputStream;
2. **import** java.io.IOException;
3. **import** java.io.PushbackInputStream;
5. /\*\*
6. \* 回退流操作
7. \* \*/
8. **public** **class** PushBackInputStreamDemo{
9. **public** **static** **void** main(String[] args) **throws** IOException{
10. String str = "hello,rollenholt";
11. PushbackInputStream push = **null**;
12. ByteArrayInputStream bat = **null**;
13. bat = **new** ByteArrayInputStream(str.getBytes());
14. push = **new** PushbackInputStream(bat);
15. **int** temp = 0;
16. **while**((temp = push.read()) != -1){
17. **if**(temp == ','){
18. push.unread(temp);
19. temp = push.read();
20. System.out.print("(回退" + (**char**) temp + ") ");
21. }**else**{
22. System.out.print((**char**) temp);
23. }
24. }
25. }
26. }

hello(回退,)rollenholt

1. /\*\*
2. \* 取得本地的默认编码
3. \* \*/
4. **public** **class** CharSetDemo{
5. **public** **static** **void** main(String[] args){
6. System.out.println("系统默认编码为：" + System.getProperty("file.encoding"));
7. }
8. }

乱码的产生：

1. **import** java.io.File;
2. **import** java.io.FileOutputStream;
3. **import** java.io.IOException;
4. **import** java.io.OutputStream;
6. /\*\*
7. \* 乱码的产生
8. \* \*/
9. **public** **class** CharSetDemo2{
10. **public** **static** **void** main(String[] args) **throws** IOException{
11. File file = **new** File("d:" + File.separator + "hello.txt");
12. OutputStream out = **new** FileOutputStream(file);
13. **byte**[] bytes = "你好".getBytes("ISO8859-1");
14. out.write(bytes);
15. out.close();
16. }
17. }

## 对象的序列化

先让我们实现一个具有序列化能力的类吧：

1. **import** java.io.\*;
2. /\*\*
3. \* 实现具有序列化能力的类
4. \* \*/
5. **public** **class** SerializableDemo **implements** Serializable{
6. **public** SerializableDemo(){
8. }
9. **public** SerializableDemo(String name, **int** age){
10. **this**.name=name;
11. **this**.age=age;
12. }
13. @Override
14. **public** String toString(){
15. **return** "姓名："+name+"  年龄："+age;
16. }
17. **private** String name;
18. **private** **int** age;
19. }

#### 在继续将序列化之前，先将一下ObjectInputStream和ObjectOutputStream这两个类

先给一个ObjectOutputStream的例子吧：

1. **import** java.io.Serializable;
2. **import** java.io.File;
3. **import** java.io.FileOutputStream;
4. **import** java.io.IOException;
5. **import** java.io.ObjectOutputStream;
7. /\*\*
8. \* 实现具有序列化能力的类
9. \* \*/
10. **public** **class** Person **implements** Serializable{
11. **public** Person(){
13. }
15. **public** Person(String name, **int** age){
16. **this**.name = name;
17. **this**.age = age;
18. }
20. @Override
21. **public** String toString(){
22. **return** "姓名：" + name + "  年龄：" + age;
23. }
25. **private** String name;
26. **private** **int** age;
27. }
28. /\*\*
29. \* 示范ObjectOutputStream
30. \* \*/
31. **public** **class** ObjectOutputStreamDemo{
32. **public** **static** **void** main(String[] args) **throws** IOException{
33. File file = **new** File("d:" + File.separator + "hello.txt");
34. ObjectOutputStream oos = **new** ObjectOutputStream(**new** FileOutputStream(
35. file));
36. oos.writeObject(**new** Person("rollen", 20));
37. oos.close();
38. }
39. }

虽然我们不能直接查看里面的内容，但是我们可以使用ObjectＩｎｐｕｔＳｔｒｅａｍ类查看：

1. **import** java.io.File;
2. **import** java.io.FileInputStream;
3. **import** java.io.ObjectInputStream;
5. /\*\*
6. \* ObjectInputStream示范
7. \* \*/
8. **public** **class** ObjectInputStreamDemo{
9. **public** **static** **void** main(String[] args) **throws** Exception{
10. File file = **new** File("d:" + File.separator + "hello.txt");
11. ObjectInputStream input = **new** ObjectInputStream(**new** FileInputStream(
12. file));
13. Object obj = input.readObject();
14. input.close();
15. System.out.println(obj);
16. }
17. }

#### Externalizable接口

现在我们来演示一下序列化和反序列话：

1. **package** IO;
3. **import** java.io.Externalizable;
4. **import** java.io.File;
5. **import** java.io.FileInputStream;
6. **import** java.io.FileOutputStream;
7. **import** java.io.IOException;
8. **import** java.io.ObjectInput;
9. **import** java.io.ObjectInputStream;
10. **import** java.io.ObjectOutput;
11. **import** java.io.ObjectOutputStream;
13. /\*\*
14. \* 序列化和反序列化的操作
15. \* \*/
16. **public** **class** ExternalizableDemo{
17. **public** **static** **void** main(String[] args) **throws** Exception{
18. ser(); // 序列化
19. dser(); // 反序列话
20. }
22. **public** **static** **void** ser() **throws** Exception{
23. File file = **new** File("d:" + File.separator + "hello.txt");
24. ObjectOutputStream out = **new** ObjectOutputStream(**new** FileOutputStream(
25. file));
26. out.writeObject(**new** Person("rollen", 20));
27. out.close();
28. }
30. **public** **static** **void** dser() **throws** Exception{
31. File file = **new** File("d:" + File.separator + "hello.txt");
32. ObjectInputStream input = **new** ObjectInputStream(**new** FileInputStream(
33. file));
34. Object obj = input.readObject();
35. input.close();
36. System.out.println(obj);
37. }
38. }
40. **class** Person **implements** Externalizable{
41. **public** Person(){
43. }
45. **public** Person(String name, **int** age){
46. **this**.name = name;
47. **this**.age = age;
48. }
50. @Override
51. **public** String toString(){
52. **return** "姓名：" + name + "  年龄：" + age;
53. }
55. // 复写这个方法，根据需要可以保存的属性或者具体内容，在序列化的时候使用
56. @Override
57. **public** **void** writeExternal(ObjectOutput out) **throws** IOException{
58. out.writeObject(**this**.name);
59. out.writeInt(age);
60. }
62. // 复写这个方法，根据需要读取内容 反序列话的时候需要
63. @Override
64. **public** **void** readExternal(ObjectInput in) **throws** IOException,
65. ClassNotFoundException{
66. **this**.name = (String) in.readObject();
67. **this**.age = in.readInt();
68. }
70. **private** String name;
71. **private** **int** age;
72. }

Serializable接口实现的操作其实是吧一个对象中的全部属性进行序列化，当然也可以使用我们上使用是Externalizable接口以实现部分属性的序列化，但是这样的操作比较麻烦，

当我们使用Serializable接口实现序列化操作的时候，如果一个对象的某一个属性不想被序列化保存下来，那么我们可以使用transient关键字进行说明：

1. **package** IO;
3. **import** java.io.File;
4. **import** java.io.FileInputStream;
5. **import** java.io.FileOutputStream;
6. **import** java.io.ObjectInputStream;
7. **import** java.io.ObjectOutputStream;
8. **import** java.io.Serializable;
10. /\*\*
11. \* 序列化和反序列化的操作
12. \* \*/
13. **public** **class** serDemo{
14. **public** **static** **void** main(String[] args) **throws** Exception{
15. ser(); // 序列化
16. dser(); // 反序列话
17. }
19. **public** **static** **void** ser() **throws** Exception{
20. File file = **new** File("d:" + File.separator + "hello.txt");
21. ObjectOutputStream out = **new** ObjectOutputStream(**new** FileOutputStream(
22. file));
23. out.writeObject(**new** Person1("rollen", 20));
24. out.close();
25. }
27. **public** **static** **void** dser() **throws** Exception{
28. File file = **new** File("d:" + File.separator + "hello.txt");
29. ObjectInputStream input = **new** ObjectInputStream(**new** FileInputStream(
30. file));
31. Object obj = input.readObject();
32. input.close();
33. System.out.println(obj);
34. }
35. }
37. **class** Person1 **implements** Serializable{
38. **public** Person1(){
40. }
42. **public** Person1(String name, **int** age){
43. **this**.name = name;
44. **this**.age = age;
45. }
47. @Override
48. **public** String toString(){
49. **return** "姓名：" + name + "  年龄：" + age;
50. }
52. // 注意这里
53. **private** **transient** String name;
54. **private** **int** age;
55. }

最后在给一个序列化一组对象的例子吧：

1. **import** java.io.File;
2. **import** java.io.FileInputStream;
3. **import** java.io.FileOutputStream;
4. **import** java.io.ObjectInputStream;
5. **import** java.io.ObjectOutputStream;
6. **import** java.io.Serializable;
8. /\*\*
9. \* 序列化一组对象
10. \* \*/
11. **public** **class** SerDemo1{
12. **public** **static** **void** main(String[] args) **throws** Exception{
13. Student[] stu = { **new** Student("hello", 20), **new** Student("world", 30),
14. **new** Student("rollen", 40) };
15. ser(stu);
16. Object[] obj = dser();
17. **for**(**int** i = 0; i < obj.length; ++i){
18. Student s = (Student) obj[i];
19. System.out.println(s);
20. }
21. }
23. // 序列化
24. **public** **static** **void** ser(Object[] obj) **throws** Exception{
25. File file = **new** File("d:" + File.separator + "hello.txt");
26. ObjectOutputStream out = **new** ObjectOutputStream(**new** FileOutputStream(
27. file));
28. out.writeObject(obj);
29. out.close();
30. }
32. // 反序列化
33. **public** **static** Object[] dser() **throws** Exception{
34. File file = **new** File("d:" + File.separator + "hello.txt");
35. ObjectInputStream input = **new** ObjectInputStream(**new** FileInputStream(
36. file));
37. Object[] obj = (Object[]) input.readObject();
38. input.close();
39. **return** obj;
40. }
41. }
43. **class** Student **implements** Serializable{
44. **public** Student(){
46. }
48. **public** Student(String name, **int** age){
49. **this**.name = name;
50. **this**.age = age;
51. }
53. @Override
54. **public** String toString(){
55. **return** "姓名：  " + name + "  年龄：" + age;
56. }
58. **private** String name;
59. **private** **int** age;
60. }