IO操作：

·掌握文件类的使用（File）

·掌握IO中流的基本使用（字符流、字节流、内存流、打印流）

·掌握System类对IO的支持

·对象序列化的问题

IO操作可以分为三大类：

·文件操作类

·字节操作类

·字符操作类

1. 文件类的使用

File类：是IO中唯一代表磁盘本身的类，File类中定义了一些与平台无关性的操作文件的方法，比如删除和创建文件，删除和创建文件夹，判断文件是会否存在等等这些方法。

构造方法：public File(String pathname)通过制定的路径去实例化一个File对象

常用的方法：

·public boolean createNewFile() throws IOException：创建文件

·public boolean delete()：删除文件

·public boolean exists()：判断文件是否存在  
 ·public boolean isDirectory()：判断此文件路径是否是文件夹

·public String[] list()：取得当前文件夹中所有的文件和文件夹

·public boolean mkdir():创建一个文件夹

范例1：在E盘下创建一个jjm.txt文件

|  |
| --- |
| **package** com.wanczy.ioDemo;  **import** java.io.File;  **import** java.io.IOException;  **public** **class** IODemo01 {  **public** **static** **void** main(String[] args) {  File file = **new** File("E:\\jjm.txt");  **if**(file.exists()){//判断此文件是否存在  file.delete();  System.*out*.println("删除文件");  }  **try** {  file.createNewFile();//创建文件  } **catch** (IOException e) {  e.printStackTrace();  }  }  } |

以上的程序会先进行判断是否存在此文件，如果存在则删除再创建，如果不存在，直接创建。

范例2：现在已经知道可以找到我们指定的文件路径了，能不能取出E盘中所有的文件呢？如果取出的是文件夹，则再从文件夹中查找。

递归：一个方法自己去调用自己。

|  |
| --- |
| **package** com.wanczy.ioDemo;  **import** java.io.File;  **public** **class** IODemo02 {  **public** **static** **void** main(String[] args) {  *loop*("E:\\DH02\\workspace");  }  **public** **static** **void** loop(String path){  File file = **new** File(path);  String s[] = **null**;//用于接收path路径下所有的文件和文件夹  **if**(file.isDirectory()){//如果File是一个文件夹  s = file.list();//取得的是文件夹中所有文件的file  **for** (**int** i = 0; i < s.length; i++) {  *loop*(path+"\\"+s[i]);  }  }**else**{  System.*out*.println(path);  }    }  } |

1. 流的基本使用：（通过File来讲解）

·字节流和字符流（重点）

在整个IO包中，流的操作分为种：

·字节流

·字节输出流（OutputStream）

·字节输入流（InputStream）

·字符流

·字符输出流（Writer）

·字符输入流（Reader）

IO基本操作步骤：在IO的操作中，完成流的操作必须要按照以下的步骤完成：

·使用File找到一个文件

·使用字节流或者字符流的子类为OutputStream、InputStream、Writer、Reader类实例化。

·进行流的读写操作

·关闭流

字节输出流（OutputStream）：

|  |
| --- |
| public abstract class OutputStreamextends Objectimplements Closeable, Flushable |

通过此类的定义可以发现此类事字节输出的最大父类，通过类的定义可以发现此类是一个抽象类，所以实例化此类对象的时候必须要依靠其子类，如果此时想要完成文件的输出操作，就必须要使用到OutputStream的一个子类（FileOutputStream）进行对OutputStream实例化。

OutputStream中常用的操作方法：

·public void write(byte[] b) throws IOException：写入一个字节数组

·public void write(byte[] b, int off, int len) throws IOException：写入部分字节数组

·public abstract void write(int b) throws IOException：写入一个数据

范例1：向jjm.txt文件中输出一段话

|  |
| --- |
| **package** com.wanczy.ioDemo;  **import** java.io.File;  **import** java.io.FileNotFoundException;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  **import** java.io.OutputStream;  **public** **class** IODemo03 {  **public** **static** **void** main(String[] args) {  File file = **new** File("E:\\jjm.txt");  OutputStream out = **null**;  **try** {  out = **new** FileOutputStream(file);//取得一个字节文件输出流的对象  String str = "Hello World!!!";  **byte** b[] = str.getBytes();  out.write(b);//写入字节数组  } **catch** (Exception e) {  e.printStackTrace();  }**finally**{  **try** {  out.close();//关闭流  } **catch** (IOException e) {  e.printStackTrace();  }  }  }  } |

以上是将整个字节数组输出到文件，当然也可以一个字节一个字节的输出：

|  |
| --- |
| package com.wanczy.ioDemo;  import java.io.File;  import java.io.FileNotFoundException;  import java.io.FileOutputStream;  import java.io.IOException;  import java.io.OutputStream;  public class IODemo04 {  public static void main(String[] args) {  File file = new File("E:\\jjm.txt");  OutputStream out = null;  try {  out = new FileOutputStream(file);//取得一个字节文件输出流的对象  String str = "Hello World!!!";  byte b[] = str.getBytes();  for (int i = 0; i < b.length; i++) {  out.write(b[i]);//一个字节一个字节写入  }  } catch (Exception e) {  e.printStackTrace();  }finally{  try {  out.close();//关闭流  } catch (IOException e) {  e.printStackTrace();  }  }  }  } |

但是以上的程序可以发现有一个问题，每次执行完程序之后，之前所有的文件内容都会替换掉，如果所希望的是内容的追加，我们就需要去观察FileoutputStream的构造方法：

|  |
| --- |
| [**FileOutputStream**](mk:@MSITStore:C:\Documents%20and%20Settings\Administrator\桌面\JDK_API_1_6_zh_CN.CHM::/java/io/FileOutputStream.html#FileOutputStream(java.io.File))([File](mk:@MSITStore:C:\Documents%20and%20Settings\Administrator\桌面\JDK_API_1_6_zh_CN.CHM::/java/io/File.html) file)            创建一个向指定 File 对象表示的文件中写入数据的文件输出流。 |
| [**FileOutputStream**](mk:@MSITStore:C:\Documents%20and%20Settings\Administrator\桌面\JDK_API_1_6_zh_CN.CHM::/java/io/FileOutputStream.html#FileOutputStream(java.io.File, boolean))([File](mk:@MSITStore:C:\Documents%20and%20Settings\Administrator\桌面\JDK_API_1_6_zh_CN.CHM::/java/io/File.html) file, boolean append)            创建一个向指定 File 对象表示的文件中写入数据的文件输出流。可以进行内容的追加 |

|  |
| --- |
| package com.wanczy.ioDemo;  import java.io.File;  import java.io.FileOutputStream;  import java.io.IOException;  import java.io.OutputStream;  public class IODemo05 {  public static void main(String[] args) {  File file = new File("E:\\jjm.txt");  OutputStream out = null;  try {  out = new FileOutputStream(file,true);//取得一个字节文件输出流的对象,true表示的是内容是追加的  String str = "Hello World!!!";  byte b[] = str.getBytes();  out.write(b);//写入字节数组  } catch (Exception e) {  e.printStackTrace();  }finally{  try {  out.close();//关闭流  } catch (IOException e) {  e.printStackTrace();  }  }  }  } |

输出：从程序中输出到别的地方就叫做输出

输入：从别的地方输入到程序

字节输入流：InputStream

|  |
| --- |
| public abstract class InputStream extends Object implements Closeable |

通过类的定义，可以发现此类是字节输入流的最高的父类，并且此类也是一个抽象类，所以实例化的时候必须要依靠其子类。如果说现在我要从文件中读取一段内容到程序，则需要使用其子类FileInputStream。

观察FileInputStream的构造方法：public FileInputStream(File file) throws FileNotFoundException：构造一个指定文件的字节输入流对象

InputStream类中方法：

·public abstract int read() throws IOException：每次只读取一个字节

·public int read(byte[] b) throws IOException：把内容全部读取到字节数组中

·public int read(byte[] b, int off, int len) throws IOException：读取指定长度的字节

范例：从文件中读取内容到程序（全部读取）

|  |
| --- |
| package com.wanczy.ioDemo;  import java.io.File;  import java.io.FileInputStream;  import java.io.InputStream;  public class IODemo06 {  public static void main(String[] args) throws Exception{  File file = new File("E:\\jjm.txt");  InputStream in = new FileInputStream(file);  byte b[] = new byte[1024];  in.read(b);//将数据读取到字节数组中  in.close();  System.out.println(new String(b));  }  } |

以上是一种比较常见的读取形式，但是以上的代码有一个缺点，就是会受到byte数组开辟空间的限制，如果说现在想要开辟一个动态数组空间，可以根据文件的大小来决定，采用read()一个字节一个字节的读取：

范例：全部读取

|  |
| --- |
| package com.wanczy.ioDemo;  import java.io.File;  import java.io.FileInputStream;  import java.io.InputStream;  public class IODemo07 {  public static void main(String[] args) throws Exception{  File file = new File("E:\\jjm.txt");  InputStream in = new FileInputStream(file);  byte b[] = new byte[(int)file.length()];  in.read(b);//将数据读取到字节数组中  in.close();  System.out.println(new String(b));  }  } |

范例：一个字节一个字节读取：

|  |
| --- |
| package com.wanczy.ioDemo;  import java.io.File;  import java.io.FileInputStream;  import java.io.InputStream;  public class IODemo08 {  public static void main(String[] args) throws Exception{  File file = new File("E:\\jjm.txt");  InputStream in = new FileInputStream(file);  byte b[] = new byte[(int)file.length()];  for (int i = 0; i < b.length; i++) {  b[i] = (byte)in.read();//进行一个字节一个字节的读取  }  System.out.println(new String(b));  }  } |

未完待续。。。

练习：做一个文件复制的功能。

·字符输出流：Writer是IO操作包中字符输出的最高父类，

|  |
| --- |
| public abstract class Writerextends Objectimplements Appendable, loseable, Flushable |

通过其类的定义发现此类是一个抽象类，所以实例化的时候必须要依靠其子类，如果现在要使用字符输出流输出数据到文件，那么就需要使用Writer的子类FileWriter。

public void write(String str) throws IOException：字符串输出

观察一下FileWriter的构造方法：public FileWriter(File file) throws IOException

|  |
| --- |
| package com.wanczy.ioDemo;  import java.io.File;  import java.io.FileWriter;  import java.io.Writer;  public class IODemo09 {  public static void main(String[] args) throws Exception{  File file = new File("E:\\jjm.txt");  Writer out = new FileWriter(file);  out.write("Hello World!!");  out.close();  }  } |

但是这个程序仍然是有问题，就是内容不能追加，那么我们怎么进行追加呢，我们继续观察FileWriter的构造方法，public FileWriter(File file, boolean append) throws IOException，此构造方法如果将boolean参数设置为true表示内容追加。

|  |
| --- |
| package com.wanczy.ioDemo;  import java.io.File;  import java.io.FileWriter;  import java.io.Writer;  public class IODemo09 {  public static void main(String[] args) throws Exception{  File file = new File("E:\\jjm.txt");  Writer out = new FileWriter(file,true);//表示内容能够追加  out.write("Hello World!!");  out.close();  }  } |

·字符输入流：Reader是一个字符输入的最高父类，

|  |
| --- |
| public abstract class Readerextends Objectimplements Readable, loseable |

发现此类仍然是一个抽象类，所以实例化的时候必须依靠其子类，如果现在我想要从文件中将内容输入到程序中，则就可以使用Reader 的子类FileReader，先观察FileReader的构造方法：

|  |
| --- |
| public FileReader(File file) throws FileNotFoundException |

发现此构造方法仍然是只需要文件路径一个参数就可以了。程序写之前我们要观察Reader中的方法：

·读取一组字符：public int read(char[] cbuf) throws IOException

·读取一个字符：public int read() throws IOException

|  |
| --- |
| package com.wanczy.ioDemo;  import java.io.File;  import java.io.FileReader;  import java.io.Reader;  public class IODemo10 {  public static void main(String[] args) throws Exception{  File file = new File("E:\\jjm.txt");  Reader in = new FileReader(file);  char c[] = new char[(int)file.length()];  in.read(c);  System.out.println(new String(c));  in.close();  }  } |

上面的程序完成一个字符数组的输入，现在使用一个字符一个字符的进行读取数据。

|  |
| --- |
| package com.wanczy.ioDemo;  import java.io.File;  import java.io.FileReader;  import java.io.Reader;  public class IODemo11 {  public static void main(String[] args) throws Exception{  File file = new File("E:\\jjm.txt");  Reader in = new FileReader(file);  char c[] = new char[(int)file.length()];  for (int i = 0; i < c.length; i++) {  c[i] = (char)in.read();  }  System.out.println(new String(c));  in.close();  }  } |

现在我们发现字节流和字符流的操作非常类似，所有的操作都可以分成两组，那么在实际开发中使用哪一组会更好呢？为了说明这个问题，我们写两个简单的程序，一个是使用字节流向文件中输入，一个是使用字符流向文件输出

·使用OutputStream

|  |
| --- |
| package com.wanczy.ioDemo;  import java.io.File;  import java.io.FileOutputStream;  import java.io.OutputStream;  public class IODemo12 {  public static void main(String[] args) throws Exception {  File file = new File("E:\\jjm.txt");  OutputStream out = new FileOutputStream(file, true);// 取得一个字节文件输出流的对象,true表示的是内容是追加的  String str = "Hello World!!!";  byte b[] = str.getBytes();  out.write(b);// 写入字节数组  // out.close();// 关闭流  }  } |

发现以上的程序没有关闭流的操作，发现数据正常输出到文件。

·字符输出流：Writer

|  |
| --- |
| package com.wanczy.ioDemo;  import java.io.File;  import java.io.FileWriter;  import java.io.Writer;  public class IODemo13 {  public static void main(String[] args) throws Exception{  File file = new File("E:\\jjm.txt");  Writer out = new FileWriter(file,true);//表示内容能够追加  out.write("Hello World!!");  // out.close();  }  } |

|  |
| --- |
|  |

以上的程序也没有关闭字符输出流，但是执行之后，内容没有向文件中输出，现在我们来使用Writer中的另外一个方法：public void flush() throws IOException刷新。

|  |
| --- |
| package com.wanczy.ioDemo;  import java.io.File;  import java.io.FileWriter;  import java.io.Writer;  public class IODemo13 {  public static void main(String[] args) throws Exception{  File file = new File("E:\\jjm.txt");  Writer out = new FileWriter(file,true);//表示内容能够追加  out.write("Hello World!!");  // out.close();  out.flush();  }  } |

从实际上来讲，在最早的程序中，并没有刷新，而只有关闭，所以表示在关闭的时候都会强制刷新，刷新的是缓冲区（内存）。

得出结论：

字节流在操作的时候，字节流是直接操作文件的，而不是操作缓冲区

·字节——文件

字符流在操作的时候，是通过缓冲区和文件间接操作的

·字符——缓冲区——文件

所以综合起来比较，在文件保存内容的时候是以字节的形式存在的，所以字节流使用比较多，但是在处理中文的时候，字符流使用比较多。

·内存操作流：如果说有临时的信息要求通过IO进行操作的话，将这些临时的信息放入到文件的话，操作步骤就比较复杂，所以此时IO提供了内存操作流，是以内存为目标进行流的输入和输出操作。

ByteArrayInputStream：将内存中的数据读取到程序中。

ByteArrayOutputStream：将程序中的数据输出到内存中。

范例：使用内存操作流完成字符串小写变大写

|  |
| --- |
| **package** com.wanczy.ioDemo;  **import** java.io.ByteArrayInputStream;  **import** java.io.ByteArrayOutputStream;  **import** java.io.InputStream;  **import** java.io.OutputStream;  **public** **class** IODemo14 {  **public** **static** **void** main(String[] args) **throws** Exception{  String str = "abc";  OutputStream out = **new** ByteArrayOutputStream();//内存输出流  InputStream in = **new** ByteArrayInputStream(str.getBytes());//内存输入流  **int** temp = 0;  **while**((temp=in.read())!=-1){ //不等于-1表示继续有数据可以读取，并且将读取的内容赋值给temp  **char** c = (**char**)temp;  out.write(Character.*toUpperCase*(c));//将字符变成大写，并且输出到内存  }  System.*out*.println(out.toString());  }  } |

·打印流：

|  |
| --- |
| public class PrintStream extends FilterOutputStream implements Appendable, Closeable |

通过类的定义可以发现打印流是字节输出流的子类的子类，我们再观察PrintStream的构造方法：

|  |
| --- |
| public PrintStream(OutputStream out) |

发现此类构造方法中的参数是父类的父类的对象，其实在java中，这种形式叫做装饰模式，可以根据构造方法中参数OutputStream的对象的不同，而向不同的地方进行输出。

范例：向文件在中输出

|  |
| --- |
| **package** com.wanczy.ioDemo;  **import** java.io.File;  **import** java.io.FileOutputStream;  **import** java.io.OutputStream;  **import** java.io.PrintStream;  **public** **class** IODemo15 {  **public** **static** **void** main(String[] args) **throws** Exception{  File file = **new** File("E:\\jjm.txt");  OutputStream os = **new** FileOutputStream(file);//这是实例化OutputStream对象  PrintStream out = **new** PrintStream(os);//这是实例化PrintStream对象  out.println("中国人民万岁");  out.print("世界人民大团结万岁");  out.close();  }  } |

得出一个结论，使用打印流可以非常方便的进行输出，以后在进行输出操作的时候，使用打印流就可以了。

1. System类对IO的支持

在System类中有三个常量：

·in ：标准的键盘输入

·out：标准的输出（输出的位置是控制台）

·err：标准的错误输出，输出的位置也是控制台

System.out:

通过API可以发现，System.out是PrintStream的对象，就是说System.out可以直接为PrintStream实例化。

|  |
| --- |
| **package** com.wanczy.ioDemo;  **import** java.io.PrintStream;  **public** **class** IODemo16 {  **public** **static** **void** main(String[] args) **throws** Exception{  PrintStream out = System.*out*;  out.print("Hello World");  out.close();  }  } |

System.err:

通过API发现，System.err是PrintStream的对象，可以直接为PrintStream实例化。

|  |
| --- |
| **package** com.wanczy.ioDemo;  **import** java.io.PrintStream;  **public** **class** IODemo17 {  **public** **static** **void** main(String[] args) **throws** Exception{  PrintStream out = System.*err*;  out.print("Hello World");  out.close();  }  } |

和System。out的效果是一样的，而且只有在Eclipse中使用System.err才会打印出红色的字体，只能从概念上去区别System。out和System。err

·System.out:一般信息的输出，可以展示给客户看

·System.err:表示的是错误的输出，而且输出的错误我们不希望用户看到

System.in:表示的使用一个标准的键盘输入，可以完成键盘输入信息的功能，而且发现System.in是InputStream的对象。可以直接为InputStream实例化。

|  |
| --- |
| **package** com.wanczy.ioDemo;  **import** java.io.InputStream;  **public** **class** IODemo18 {  **public** **static** **void** main(String[] args)**throws** Exception {  InputStream in = System.*in*;  **byte**[] b = **new** **byte**[1024];//用于接收字节输入流  System.*out*.print("请输入内容：");  **int** length = in.read(b);//进行输入内容的读取  System.*out*.println("输入的内容为：" + **new** String(b,0,length));  }  } |

以上的程序已经完成了键盘的输入，但是这个程序有一个问题，就是受到字节数组长度的限制，而且在输入中文的时候也会存在问题，此时可以通过另外一种方式，不指定大小，边读边判断。

|  |
| --- |
| **package** com.wanczy.ioDemo;  **import** java.io.InputStream;  **public** **class** IODemo19 {  **public** **static** **void** main(String[] args)**throws** Exception {  InputStream in = System.*in*;  System.*out*.print("请输入内容：");  StringBuffer sb = **new** StringBuffer();  **int** temp = 0;  **while**((temp = in.read())!=-1){//不等于-1表示还没有读取到最后一个，  **char** c = (**char**)temp;  **if**(c == '\n'){  **break**;  }  sb.append(c);  }  System.*out*.println("输入的内容为：" + sb.toString());  }  } |

这个时候我们完成了输入时一个字节一个字节的读取，并且没有受到数组长度的限制，但是此时如果输入的是中文的话，就无法正常的读取，因为一个中文有两个字节组成，如果现在出现一个方法，一次性能读取一行的话，那么输入中文乱码的问题就可以解决，这个时候我们需要使用到BufferedReader类完成。

BufferedReader:

观察此类的构造方法：public BufferedReader(Reader in)发现如果想要使用BufferedReader的话就必须要使用Reader对象，但是发现Reader对象无法由System.in直接生成，因为System.in是Inputstream的类型。

在java中提供了专门的两个类，用于字节——字符的转换。

·InputStreamReader:表示将字节输入流转换成字符输入流

·OutputStreamWriter：表示将字节输出变为字符输出

现在我们要实现输入能够进行一行一行读取的话，是要将字节输入流转换成字符输入流，所以要使用到的是InputStreamReader，现在我们来观察此类。

·观察此类构造方法：public InputStreamReader(InputStream in)可以由一个InputStream的对象直接实例化InputStreamReader对象。

·观察此类的结构：public class InputStreamReader extends Reader：发现这个类是Reader的子类

在继续观察BufferedReader的方法：public String readLine() throws IOException：一次性读取一行的方法。

|  |
| --- |
| **package** com.wanczy.ioDemo;  **import** java.io.BufferedReader;  **import** java.io.InputStream;  **import** java.io.InputStreamReader;  **import** java.io.Reader;  **public** **class** IODemo20 {  **public** **static** **void** main(String[] args) **throws** Exception {  InputStream in = System.*in*;  Reader isr = **new** InputStreamReader(in);  BufferedReader br = **new** BufferedReader(isr);  System.*out*.print("请输入内容");  String str = br.readLine();//一次性读取输入的一行内容  System.*out*.println("输入的内容为：" + str);  }  } |

现在已经完成一次性读取输入一行内容的程序，解决了中文乱码的问题，这是最合适的，也是java中键盘输入数据的标准格式。

此程序可以简写：

|  |
| --- |
| **package** com.wanczy.ioDemo;  **import** java.io.BufferedReader;  **import** java.io.InputStreamReader;  **public** **class** IODemo20 {  **public** **static** **void** main(String[] args) **throws** Exception {  BufferedReader br = **new** BufferedReader(**new** InputStreamReader(System.*in*));  System.*out*.print("请输入内容");  String str = br.readLine();//一次性读取输入的一行内容  System.*out*.println("输入的内容为：" + str);  }  } |

·输入输出重定向：我们知道System.in和System.out都是有固定的输入输出目标，但是System类中定义了输入输出重定向的方法，用于从不同地方进行输入输出。

·输入重定向：public static void setIn(InputStream in)

·输出重定向：public static void setOut(PrintStream out)

输出重定向：

|  |
| --- |
| **package** com.wanczy.ioDemo;  **import** java.io.File;  **import** java.io.PrintStream;  **public** **class** IODemo21 {  **public** **static** **void** main(String[] args) **throws** Exception {  File file = **new** File("E:\\jjm.txt");  System.*setOut*(**new** PrintStream(file));  System.*out*.println("你好吗，我很好");  }  } |

输入重定向：

|  |
| --- |
| package com.wanczy.ioDemo;  import java.io.BufferedReader;  import java.io.File;  import java.io.FileInputStream;  import java.io.InputStreamReader;  public class IODemo22 {  public static void main(String[] args) throws Exception {  File file = new File("E:\\jjm.txt");  System.setIn(new FileInputStream(file));  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  String str = br.readLine();//一次性读取输入的一行内容  System.out.println("输入的内容为：" + str);  }  } |

System.in一般不会去修改输入方法，我们一般修改的是System.err,因为我们要将错误输入到日志文件中。

1. 对象的序列化：将对象写入到流中，

对象的反序列化：从流中恢复对象。

对象序列化的好处就是即使程序终止运行，对象依然可以保留，如果要进行对象序列化的话，这个对象的类必须要实现java序列化的接口Serializable。

对象序列化的步骤：

·要创建一个对象输出流：ObjectOutputStream，可以连接到其他的流，比如说文件输出流。

观察此类的构造方法：public ObjectOutputStream(OutputStream out) throws IOException

观察ObjectOutputStream里面的方法：public final void writeObject(Object obj) throws IOException将一个对象放入到对象输出流中。

·关闭流。

|  |
| --- |
| **package** com.wanczy.ioDemo;  **import** java.io.File;  **import** java.io.FileOutputStream;  **import** java.io.ObjectOutputStream;  **public** **class** IODemo23 {  **public** **static** **void** main(String[] args) **throws** Exception {  Person p = **new** Person(20,"黄春林");  File file = **new** File("E:\\jjm.txt");  ObjectOutputStream oos = **new** ObjectOutputStream(**new** FileOutputStream(file));  oos.writeObject(p);//将对象输出流写入到文件输出流中  oos.close();  }  } |

以上已经将对象保留到文件中了。

对象的反序列化：

·ObjectInputStream：对象输入流

·readObject():从流中读取对象的方法

|  |
| --- |
| package com.wanczy.ioDemo;  import java.io.File;  import java.io.FileInputStream;  import java.io.ObjectInputStream;  public class IODemo24 {  public static void main(String[] args) throws Exception {  File file = new File("E:\\jjm.txt");  ObjectInputStream oos = new ObjectInputStream(new FileInputStream(file));  Person p = (Person)oos.readObject();//将流中的对象读取到程序中  System.out.println(p.getName()+"\t" + p.getAge());  oos.close();  }  } |

如果说不希望类中的某个属性进行序列化的话，则使用transient关键字去修饰属性，表示此属性不进行序列化.

总结IO:所有内容都是重点