Java SE程序设计 北京圣思园科技有限公司

主讲人　张龙

Java I/O系统

- 课程目标
  - 理解Java I/O系统
  - 熟练使用java.io包中的相关类与接口进行I/O编程
  - 掌握Java I/O的设计原则与使用的设计模式

字符流

- 尽管字节流提供了处理任何类型输入/输出操作的足够的功能，它们不能直接操作Unicode字符。既然Java的一个主要目的是支持“只写一次，到处运行”的哲学，包括直接的字符输入 / 输出支持是必要的。本节将讨论几个字符输入 / 输出类。字符流层次结构的顶层是Reader 和Writer 抽象类。我们将从它们开始

字符流

- 字符输入/输出类是在java 的1.1版本中新加的。由此，你仍然可以发现遗留下的程序代码在应该使用字符流时却使用了字节流。当遇到这种代码，最好更新它

字符流Reader和Writer类

- 由于Java采用16位的Unicode字符，因此需要基于字符的输入/输出操作。从Java1.1版开始，加入了专门处理字符流的抽象类Reader和Writer，前者用于处理输入，后者用于处理输出。这两个类类似于InputStream和OuputStream，也只是提供一些用于字符流的规定，本身不能用来生成对象

字符流Reader和Writer类

- Reader和Writer类也有较多的子类，与字节流类似，它们用来创建具体的字符流对象进行I/O操作。字符流的读写等方法与字节流的相应方法都很类似，但读写对象使用的是字符

字符流Reader和Writer类

- Reader中包含一套字符输入流需要的方法，可以完成最基本的从输入流读入数据的功能。当Java程序需要外设的数据时，可根据数据的不同形式，创建一个适当的Reader子类类型的对象来完成与该外设的连接，然后再调用执行这个流类对象的特定输入方法，如read()，来实现对相应外设的输入操作

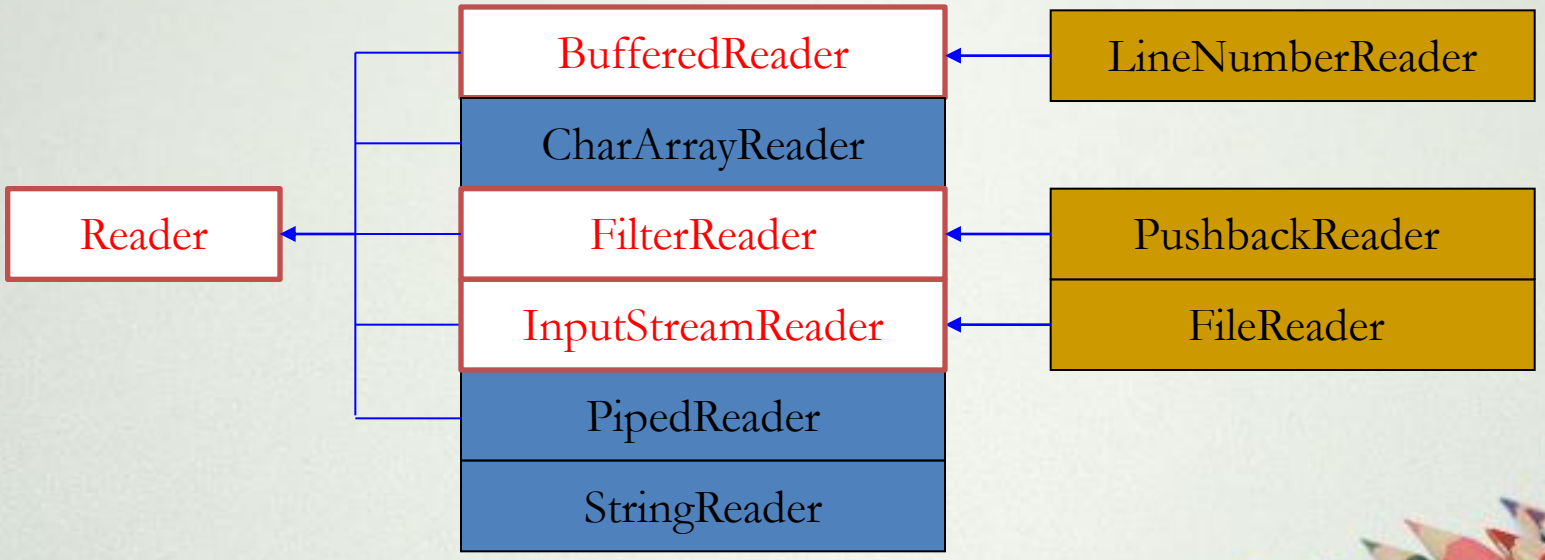
字符流Reader和Writer类

- Writer中包含一套字符输出流需要的方法，可以完成最基本的输出数据到输出流的功能。当Java程序需要将数据输出到外设时，可根据数据的不同形式，也要创建一个适当的Writer子类类型的对象来完成与该外设的连接，然后再调用执行这个流类对象的特定输出方法，如write()，来实现对相应外设的输出操作

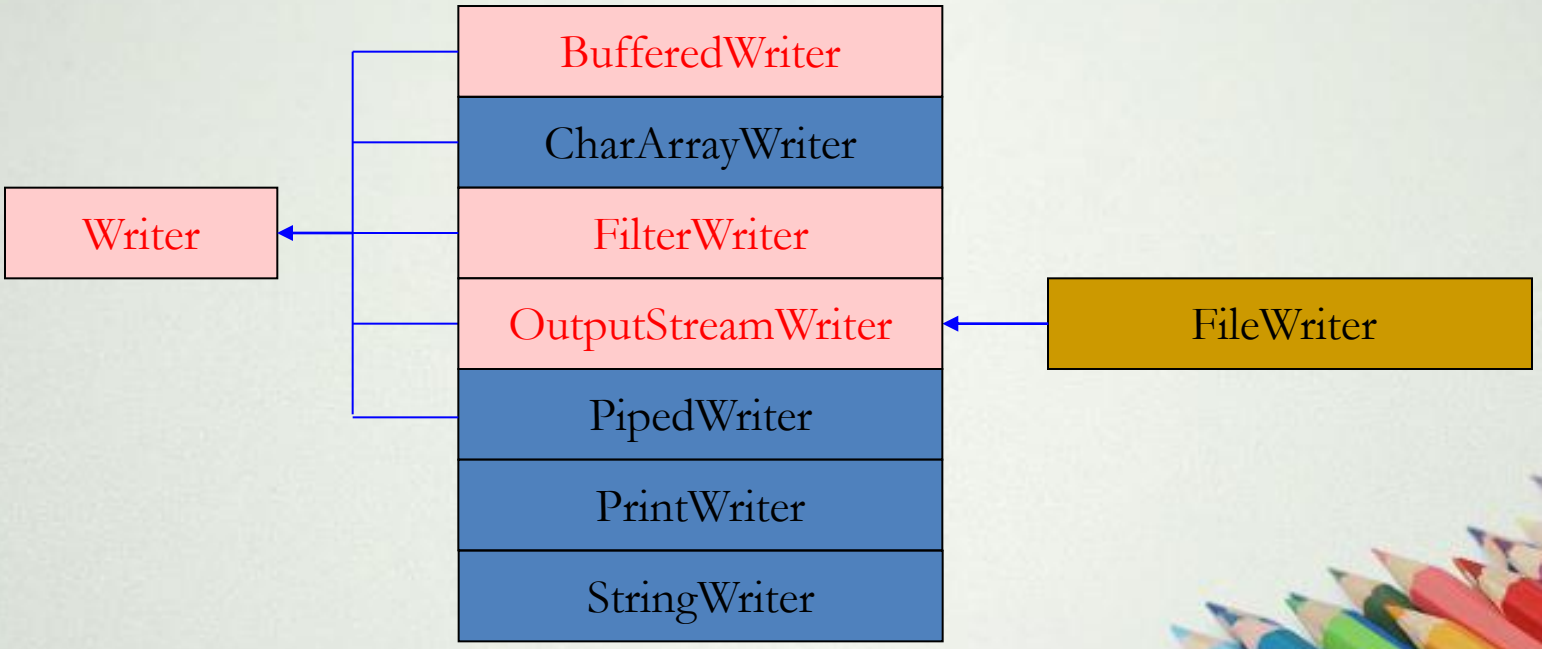
字符流Reader和Writer类

- Reader是定义Java的流式字符输入模式的抽象类。该类的所有方法在出错情况下都将引发IOException 异常
- Writer 是定义流式字符输出的抽象类。所有该类的方法都返回一个void 值并在出错条件下引发IOException 异常

java.io包中Reader的类层次



java.io包中Writer的类层次



字符流Reader和Writer类

- Java程序语言使用Unicode来表示字符串和字符，Unicode使用两个字节来表示一个字符，即一个字符占16位

InputStreamReader和OutputStreamWriter类

- 这是java.io包中用于处理字符流的基本类，用来在字节流和字符流之间搭一座“桥”。这里字节流的编码规范与具体的平台有关，可以在构造流对象时指定规范，也可以使用当前平台的缺省规范

InputStreamReader和OutputStreamWriter类

- InputStreamReader和OutputStreamWriter类的主要构造方法如下
  - public InputSteamReader(InputSteam in)
  - public InputSteamReader(InputSteam in,String enc)
  - public OutputStreamWriter(OutputStream out)
  - public OutputStreamWriter(OutputStream out,String enc)

InputStreamReader和OutputStreamWriter类

- 其中in和out分别为输入和输出字节流对象，enc为指定的编码规范（若无此参数，表示使用当前平台的缺省规范，可用getEncoding()方法得到当前字符流所用的编码方式）。
- 读写字符的方法read()、write()，关闭流的方法close()等与Reader和Writer类的同名方法用法都是类似的。

InputStreamReader和OutputStreamWriter类

- 参见程序 StreamTest.java
  - 该程序将两行字符串写入文本中，并从中读取出来显示在命令行上
- 参见程序 StreamTest2.java
  - 该程序将来自标准输入的字符串显示在标准输出上

FileReader

- FileReader类创建了一个可以读取文件内容的Reader类。FileReader继承于InputStreamReader。它最常用的构造方法显示如下
  - FileReader(String filePath)
  - FileReader(File fileObj)
  - 每一个都能引发一个FileNotFoundException异常。这里，filePath是一个文件的完整路径，fileObj是描述该文件的File 对象

FileReader

- 参见程序 FileReader1.java
  - 该例子演示了怎样从一个文件逐行读取并把它输出到标准输出流。例子读它自己的源文件。

FileWriter

- FileWriter 创建一个可以写文件的Writer类。 FileWriter继承于OutputStreamWriter.它最常用的构造方法如下：
  - FileWriter(String filePath)
  - FileWriter(String filePath, boolean append)
  - FileWriter(File fileObj)
  - append ：如果为 true，则将字节写入文件末尾处，而不是写入文件开始处

FileWriter

- 它们可以引发IOException或SecurityException异常。这里，filePath是文件的完全路径，fileObj是描述该文件的File对象。如果append为true，输出是附加到文件尾的。
- FileWriter类的创建不依赖于文件存在与否。在创建文件之前，FileWriter将在创建对象时打开它来作为输出。如果你试图打开一个只读文件，将引发一个IOException异常

FileWriter

- 参见程序 FileWriter1.java
  - 该例子是前面讨论FileOutputStream时用到例子的字符流形式的版本
  - 它创建了一个样本字符缓冲器，开始生成一个String，然后用getChars( )方法提取字符数组。然后该例创建了三个文件。第一个file1.txt，包含例子中的偶数个字符。第二个file2.txt，包含所有的字符。最后，第三个文件file3.txt，只含有最后的四分之一

CharArrayReader

- CharArrayReader 是一个把字符数组作为源的输入流的实现。该类有两个构造方法，每一个都需要一个字符数组提供数据源
  - CharArrayReader(char array[ ])
  - CharArrayReader(char array[ ], int start, int numChars)
  - 这里，array是输入源。第二个构造方法从你的字符数组的子集创建了一个Reader，该子集以start指定的索引开始，长度为numChars

CharArrayReader

- 参见程序 CharArrayReader1.java
  - 该例子用到了上述CharArrayReader的两个构造方法
  - public void **getChars**(int srcBegin, int srcEnd, char[] dst, int dstBegin)
    - 将字符从此字符串复制到目标字符数组。 要复制的第一个字符在索引 srcBegin 处；要复制的最后一个字符在索引 srcEnd-1 处（因此要复制的字符总数是 srcEnd-srcBegin）。要复制到 dst 子数组的字符从索引 dstBegin 处开始，并结束于索引：

      dstbegin + (srcEnd-srcBegin) - 1

CharArrayWriter

- CharArrayWriter 实现了以数组作为目标的输出流。CharArrayWriter 有两个构造方法
  - CharArrayWriter( )
  - CharArrayWriter(int numChars)
  - 第一种形式，创建了一个默认长度的缓冲区。
  - 第二种形式，缓冲区长度由numChars指定。缓冲区保存在CharArrayWriter的buf 成员中。缓冲区大小在需要的情况下可以自动增长。缓冲区保持的字符数包含在CharArrayWriter的count 成员中。buf 和count 都是受保护的域(protected)

CharArrayWriter

- 参见程序 CharArrayWriter1.java
  - 我们继续使用前面显示的 ByteArrayOutputStream 例子中演示的程序

BufferedReader

- BufferedReader 通过缓冲输入提高性能。它有两个构造方法
  - `BufferedReader(Reader inputStream)`
  - BufferedReader(Reader inputStream, int bufSize)
  - 第一种形式创建一个默认缓冲区长度的缓冲字符流。第二种形式，缓冲区长度由bufSize传入
- 和字节流的情况相同，缓冲一个输入字符流同样提供支持可用缓冲区中流内反向移动的基础。为支持这点，BufferedReader 实现了mark( )和reset( )方法，并且BufferedReader.markSupported( ) 返回true

BufferedReader

- 参见程序 BufferedReader1.java
  - 该程序实现了之前用BufferedInputStream实现的同样的程序功能

BufferedWriter

- BufferedWriter是一个增加了flush( )方法的Writer。flush( )方法可以用来确保数据缓冲区确实被写到实际的输出流。用BufferedWriter 可以通过减小数据被实际的写到输出流的次数而提高程序的性能。

BufferedWriter

- BufferedWriter有两个构造方法：
  - BufferedWriter(Writer outputStream)
  - BufferedWriter(Writer outputStream, int bufSize)
  - 第一种形式创建了使用默认大小缓冲区的缓冲流。第二种形式中，缓冲区大小是由bufSize参数传入的

PushbackReader

- PushbackReader类允许一个或多个字符被送回输入流。这使你可以对输入流进行预测。下面是它的两个构造方法
  - PushbackReader(Reader inputStream)
  - PushbackReader(Reader inputStream, int bufSize)
  - 第一种形式创建了一个允许单个字节被推回的缓冲流。
  - 第二种形式，推回缓冲区的大小由bufSize参数传入

PushbackReader

- PushbackReader 提供了unread( )方法。该方法返回一个或多个字符到调用的输入流。它有下面的三种形式
  - void unread(int ch)
  - void unread(char buffer[ ])
  - void unread(char buffer[ ], int offset, int numChars)
  - 第一种形式推回ch传入的字符。它是被并发调用的read( )返回的下一个字符。第二种形式返回buffer中的字符。第三种形式推回buffer中从offset开始的numChars个字符。如果在推回缓冲区为满的条件下试图返回一个字符，一个IOException异常将被引发

PushbackReader

- 参见程序 PushbackReader1.java
  - 该例子重写了前面的PushBackInputStream例子，用PushbackReader代替了PushBackInputStream。和以前一样，它演示了一个编程语言解析器怎样用一个推回流处理用于比较的==操作符和用于赋值的=操作符之间的不同

字符集的编码

- ASCII(American Standard Code for Information Interchange，美国信息互换标准代码)，是基于常用的英文字符的一套电脑编码系统。我们知道英文中经常使用的字符、数字符号被计算机处理时都是以二进制码的形式出现的。这种二进制码的集合就是所谓的ASCII码。每一个ASCII码与一个8位（bit）二进制数对应。其最高位是0，相应的十进制数是0-127。如，数字“0”的编码用十进制数表示就是48。另有128个扩展的ASCII码，最高位都是1，由一些制表符和其它符号组成。ASCII是现今最通用的单字节编码系统。
- GB2312：GB2312码是中华人民共和国国家汉字信息交换用编码，全称《信息交换用汉字编码字符集－基本集》。主要用于给每一个中文字符指定相应的数字，也就是进行编码。一个中文字符用两个字节的数字来表示，为了和ASCII码有所区别，将中文字符每一个字节的最高位置都用1来表示。

字符集的编码

- GBK：为了对更多的字符进行编码，国家又发布了新的编码系统GBK(GBK的K是“扩展”的汉语拼音第一个字母)。在新的编码系统里，除了完全兼容GB2312 外，还对繁体中文、一些不常用的汉字和许多符号进行了编码。
- ISO-8859-1：是西方国家所使用的字符编码集，是一种单字节的字符集 ，而英文实际上只用了其中数字小于128的部分。

字符集的编码

- Unicode：这是一种通用的字符集，对所有语言的文字进行了统一编码，对每一个字符都用2个字节来表示，对于英文字符采取前面加“0”字节的策略实现等长兼容。如 “a” 的ASCII码为0x61，UNICODE就为0x00，0x61。(在internet上传输效率较低)
- UTF-8：Eight-bit UCS Transformation Format，(**UCS，Universal Character Set，通用字符集，UCS 是所有其他字符集标准的一个超集**)。一个7位的ASCII码值，对应的UTF码是一个字节。如果字符是0x0000，或在0x0080与0x007f之间，对应的UTF码是两个字节，如果字符在0x0800与0xffff之间，对应的UTF码是三个字节(汉字为3个字节)。

字符集的编码

- 参见程序CharSet1.java
  - 该程序返回了在当前系统中所有可用的字符集
- 参见程序CharSet2.java
  - 我们可以查看到输出该行结果：file.encoding=GBK，这说明在该系统上采用的字符编码方式为GBK

RandomAccessFile（随机访问文件类）

- RandomAccessFile包装了一个随机访问的文件。它不是派生于InputStream和OutputStream，而是实现定义了基本输入/输出方法的DataInput和DataOutput接口。它支持定位请求——也就是说，可以在文件内部放置文件指针。它有两个构造方法：

RandomAccessFile（随机访问文件类）

- RandomAccessFile(File fileObj, String access) throws FileNotFoundException
- RandomAccessFile(String filename, String access) throws FileNotFoundException
- 第一种形式，fileObj指定了作为File 对象打开的文件的名称。
- 第二种形式，文件名是由filename参数传入的。
- 两种情况下，access 都决定允许访问何种文件类型。如果是 “r”，那么文件可读不可写，如果是 “rw”，文件以读写模式打开

RandomAccessFile（随机访问文件类）

例如：

new RandomAccessFile(“test.txt", "r");

new RandomAccessFile(“test.txt", "rw");

RandomAccessFile（随机访问文件类）

- RandomAccessFile类同时实现了DataInput和DataOutput接口，提供了对文件随机存取的功能，利用这个类可以在文件的任何位置读取或写入数据。
- RandomAccessFile类提供了一个文件指针，用来标志要进行读写操作的下一数据的位置。

RandomAccessFile（随机访问文件类）

- 常用方法：
  - **public long getFilePointer()**
  - 返回到此文件开头的偏移量（以字节为单位），在该位置发生下一个读取或写入操作
  - **public void seek(long pos)**
  - 设置到此文件开头测量到的文件指针偏移量，在该位置发生下一个读取或写入操作。偏移量的设置可能会超出文件末尾。偏移量的设置超出文件末尾不会改变文件的长度。只有在偏移量的设置超出文件末尾的情况下对文件进行写入才会更改其长度

RandomAccessFile（随机访问文件类）

- 常用方法：
  - public long **length**()
  - 返回此文件的长度
  - public int **skipBytes**(int n)
  - 尝试跳过输入的 n 个字节以丢弃跳过的字节

RandomAccessFile（随机访问文件类）

- 参见程序 RandomAccessFile1.java
  - RandomAccessFile将文件的读写放置在一个类中，因此对于文件的读写是相当的方便的
- 参见程序 RandomAccessFile2.java
  - 该程序的功能与上一个类似，同样是先写入文件中数据，然后读取出来，需要注意判断文件中数据的精确位置，否则将会出错

序列化

- 将对象转换为字节流保存起来，并在以后还原这个对象，这种机制叫做对象序列化。
- 将一个对象保存到永久存储设备上称为持久化。
- 一个对象要想能够实现序列化，必须实现Serializable接口或Externalizable接口。

序列化

- 序列化（serialization）是把一个对象的状态写入一个字节流的过程。当你想要把你的程序状态存到一个固定的存储区域例如文件时，它是很管用的。稍后一点时间，你就可以运用序列化过程存储这些对象

序列化

- 假设一个被序列化的对象引用了其他对象，同样，其他对象又引用了更多的对象。这一系列的对象和它们的关系形成了一个顺序图表。在这个对象图表中也有循环引用。也就是说，对象X可以含有一个对象Y的引用，对象Y同样可以包含一个对象X的引用。对象同样可以包含它们自己的引用。对象序列化和反序列化的工具被设计出来并在这一假定条件下运行良好。如果你试图序列化一个对象图表中顶层的对象，所有的其他的引用对象都被循环的定位和序列化。同样，在反序列化过程中，所有的这些对象以及它们的引用都被正确的恢复

序列化

- 当一个对象被序列化时，只保存对象的非静态成员变量，不能保存任何的成员方法和静态的成员变量。
- 如果一个对象的成员变量是一个对象，那么这个对象的数据成员也会被保存。
- 如果一个可序列化的对象包含对某个不可序列化的对象的引用，那么整个序列化操作将会失败，并且会抛出一个NotSerializableException。我们可以将这个引用标记为transient，那么对象仍然可以序列化

序列化

- Serializable接口
  - 只有一个实现Serializable接口的对象可以被序列化工具存储和恢复。Serializable接口没有定义任何成员。它只用来表示一个类可以被序列化。如果一个类可以序列化，它的所有子类都可以序列化。
  - 声明成transient的变量不被序列化工具存储。同样，static变量也不被存储

序列化

- Externalizable接口
- Java的序列化和反序列化的工具被设计出来，所以很多存储和恢复对象状态的工作自动进行。然而，在某些情况下，程序员必须控制这些过程。例如，在需要使用压缩或加密技术时，Externalizable接口为这些情况而设计。
- Externalizable 接口定义了两个方法：
  - void readExternal(ObjectInput inStream) throws IOException, ClassNotFoundException
  - void writeExternal(ObjectOutput outStream) throws IOException
  - 这些方法中，inStream是对象被读取的字节流，outStream是对象被写入的字节流。

序列化

- Externalizable 实例类的惟一特性是可以被写入序列化流中，该类负责保存和恢复实例内容。 若某类要完全控制某一对象及其超类型的流格式和内容，则它要实现 Externalizable 接口的 writeExternal 和 readExternal 方法。这些方法必须显式与超类型进行协调以保存其状态。这些方法将代替自定义的 writeObject 和 readObject 方法实现

序列化

- ObjectOutput接口
  - ObjectOutput 继承DataOutput接口并且支持对象序列化。特别注意writeObject( )方法，它被称为序列化一个对象。所有这些方法在出错情况下引发IOException 异常

序列化

- ObjectOutputStream类
  - ObjectOutputStream类继承OutputStream 类和实现ObjectOutput 接口。它负责向流写入对象。该类的构造方法如下：
  - ObjectOutputStream(OutputStream outStream) throws IOException
  - 参数outStream 是序列化的对象将要写入的输出流

序列化

- ObjectInput
  - ObjectInput 接口继承DataInput接口。它支持对象序列化。特别注意 readObject( )方法，它叫反序列化对象。所有这些方法在出错情况下引发IOException 异常

序列化

- ObjectInputStream
  - ObjectInputStream 继承InputStream 类并实现ObjectInput 接口。ObjectInputStream 负责从流中读取对象。该类的构造方法如下：
  - ObjectInputStream(InputStream inStream) throws IOException,StreamCorruptedException
  - 参数inStream 是序列化对象将被读取的输入流。

序列化

- 参见程序 Serializable1.java
  - 反序列化时不会调用对象的任何构造方法，仅仅根据所保存的对象状态信息，在内存中重新构建对象

序列化

- 在序列化和反序列化进程中需要特殊处理的 Serializable 类应该实现以下方法：
- private void writeObject(java.io.ObjectOutputStream stream)
  throws IOException;
- private void readObject(java.io.ObjectInputStream stream)
  throws IOException, ClassNotFoundException;

这两个方法不属于任何一个类和任何一个接口，是非常特殊的方法.

序列化

- 参见程序Serializable2.java

总结

- InputStream和OutputStream：字节流的输入输出。
- Reader和Writer：字符流的输入输出。
- 流的链接(Java I/O库的设计原则)