1. 输入与输出
2. 输入流 / 输出流

从其中读入一个字节序列的对象称为输入流，而可以向其中写入一个字节序列的对象称为输出流。这些字节序列的来源可以是文件，网络连接和内存块。抽象类InputStream和OutputStream构成了IO类层次结构的基础，但是面向字节的流无法处理Unicode形式的信息（每个Unicode字符使用了多个字节表示），所以从抽象类Reader和Writer中继承出来了用于处理Unicode字符的单独的类层次结构，这些类都是基于两个字节的Char值的。当然还有其他在处理字节基础上的更上层的类，如处理二进制的类DataInputStream, DataOutputStream，处理zip压缩文件的ZipInputStream, ZipOutputStream类等等。

1. InputStream类
2. read()

抽象方法read()，读入一个字节，返回读入的字节，在遇到结尾时返回-1。在设计具体的输入流类时，需要覆盖这个方法来提供适当的功能，执行时都会阻塞，直到字节真正被读入。

1. read(byte[] b)

读入一个字节数组，返回实际读入的字节数。

1. transferTo(OutputStream out)

将当前输入流中的所有字节传送到指定的输出流中，返回传递的字节数。

1. available()

在不阻塞的情况下可获取的字节数，可以用于检查当前可读入的字节数量，便于进行之后的操作。

1. close()

关闭这个输入流，释放操作系统资源。

1. mark(int readlimit)

在输入流的当前位置打上标记。

1. reset()

返回最后一个标记，随后对read的调用将重新读入这些字节。

1. OutputStream
2. write(int n)

抽象类，写出一个字节的数据。

1. close()

冲刷用于该输出流的缓冲区，并且关闭该输出流，所有被临时置于缓冲区中，以便用更大的包的形式传递的字节都会在关闭输出流的时候被送出，如果不关闭输出流，那么写出字节的最后一个包可能永远不会传递。

1. flush()

冲刷输出流，即将缓冲区的数据送出。

1. 文件输入输出流

FileInputStream 和 FileOutputStream可以提供附着在磁盘文件上的输入流和输出流，只需要向构造器提供文件名或文件的完整路径名。如下所示：

var fin = new FileInputStream(“a.txt”);

所有java.io中的类都将相对路径名解释为以用户工作目录开始，且由于反斜杠“\”在Java字符串中是转义字符，所以在Windows风格的路径名中使用“\\”，如C:\\Windows\\win.ini。

1. 组合过滤器

对于不同的输入流，可以嵌套在一起使用以获得更加强大灵活的功能，如从一个文件中读取输入流，并且通过缓冲机制读取，最后通过转换为二进制，就可以用以下的方式：

var din = new DataInputStream(new BufferedInputStream( new FileInputStream(“a.txt”)));

这种复杂的构造器序列可以灵活地将不同的输入流组合在一起，非常强大。

如预览下一个字节是否是期望的值，如果不是，将其推回到流中：

var pbin = new PushbackInputStream(new BufferedInputStream( new FileInputStream(“a.txt”)));

int b = pbin.read();

if( b != ‘<’) pbin.unread(b);

1. 文本输入输出

文本格式的IO适合人类阅读，在存储文本字符串时，需要考虑字符编码方式，OutputStreamWriter类将使用选定的字符编码方式，把Unocide码元的输出流转换为字节流，UTF-8是互联网上最常用的编码方式。如从控制台读入键盘敲击信息，并转换为Unicode：

var int = new InputStreamReader(System.in);

或指定编码方式：

var in = new InputStreamReader(new FileInputStream(“data.txt”, StandardCharsets.UTF\_8));

对于文本输出，可以使用PrintWriter，与System.out相同的print, println, prinf等方法，即用文本输出到文件中，如下所示：

var out = new PrintWriter(“employee.txt”, StandardCharsets.UTF\_8);

也可以嵌套输出使用：

var out = new PrintWriter(new OutputStreamWriter(new FileOutputStream(“employee.txt”, StandardCharsets.UTF\_8), true));

最简单的处理文本的方式就是使用Scanner类，我们可以从任何输入流中构建Scanner对象，扫描一行行进来。

1. 字符编码方式

Java针对字符使用的是Unicode标准，每个字符或者“编码点”都具有一个21位的整数，有多种不同的字符编码方式，UTF-8将每个Unocide编码点编码为1到4个字节的序列，包含了英语中用到的所有字符的ASICC字符集中的每个字符都只会占用一个字节，UTF-8会将每个Unicode编码点编码为1个或2个16位值，需要考虑高位低位的优先顺序，可以在文件的开头来表示使用的是哪一种格式。