# Java IO 教程

<https://www.jianshu.com/p/8dde5ac614e9>

Java IO是java中的相关API，主要目的为读数据与写数据(input 和 output)。大部分的应用都需要处理一些输入数据，并且根据输入数据生成一些输出数据，比如说从文件或网络读取数据，然后写回文件或通过网络响应数据。

Java IO的相关API在Java IO包中(java.io)。如果你直接去java.io包中去看源代码，那么你会因为大量的代码而感到相当困惑。这些Java类的意图是什么？哪些类可以用来完成你的任务？如何创建你自己的类去做插件？等等。那么这个教程的目的就是尝试去告诉你这些Java类是如何组织的和它们的一些背后目的，所以你不必去疑惑怎么样去选择使用合适的类，或者有没有一个现成的类去满足你的需求。

## java.io包的范围

java.io包并不能解决所有的input和output。实际上从GUI程序或web页面输入或输出到他们上，这类的api并没有在java.io包中，而是在另外的一些地方。比如说，Swing工程或servlet和http相关的IO类就在javaEE中。

java.io包首先是专注于解决文件、网络流、内存缓冲区等的输入或输出。然而，java.io包并不包含socket这些的必要的网络通讯，如果需要，你可以去看Java Networking API。但是你打开一个socket连接，这时候读写数据就需要用InputStream和OutputStream等相关的类。

## Java NIO - 另一种 IO API

Java也包含另一种io的API，叫做java NIO，他和Java IO与Java Networking API的java类有很多的相似之处。但是Java NIO可以以非阻塞模式来工作。非阻塞模式在高并发下读写数据的性能要远远大于堵塞的IO。

## 其他的Java IO工具和一些技巧等

可以点击教程链接[Java How To's and Utilities](http://tutorials.jenkov.com/java-howto/index.html)，其中包含了一些新的Java IO使用工具。

## 此Java IO教程的范围

首先是刚告诉你Java IO是如何功能工作的，以及告诉你如何去使用它。最后，会转向到Java IO包的核心类上。

这个教程中展示的类，不仅仅只是一个API的列表展示（你可以从java官方网站上获取这些API列表）。每段文本都是对类的简短介绍，他的目的和一些使用例子。换句话说，一些东西你没有必要去java官方文档上去找。

## Java5 到 Java8

这个教程的第一个版本是基于Java5来写的,写的时候Java已经到了Java8版本，但是这些代码一样可以在Java8上面完美运行。

# Java IO 概述

在这里，我会给你一个Java.io包中的类的一个概要说明。更具体的说，我回去尝试按照它们的意图来分组这些类。这些分组会放你未来会觉得它更简单，入去确定一个类的目的，或根据你的需要去找到合适的类。

## Input 和Output - 源代码和目标

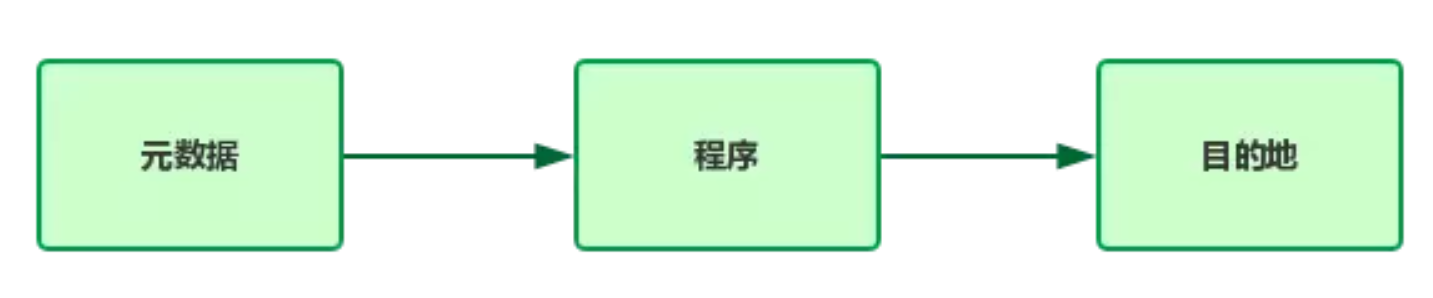
专业术语“input”和“output”有时候会有一些概念上的模糊。一个应用程序的输入部分往往是另一个部分的输出。OutputStream是一个流被写出去，还是从一个地方写进来？归根结底，OutputStream是把数据写入到读取的程序，不是么？以我个人经历发现，第一次学习Java IO的时候这些会让我有一些困惑。

为了解决这些困惑，我曾经尝试在input和output上添加一些不同的名称，希望让从哪里输入的，输出到哪里，有一些概念上的联系。

Java的IO包本身的主要关注点在于，从一个地方读取原始数据和向目的地写入数据。最典型的数据来源和目的地主要有：

* 文件（File）
* 管道（Pipes）
* 网络连接（Network Connections）
* 内存缓冲区，比如数组（In-memory Buffers）
* System.in, System.out, System.error

下面的图说明了一个程序从一个地方读数据并且写数据到目的地的原理：

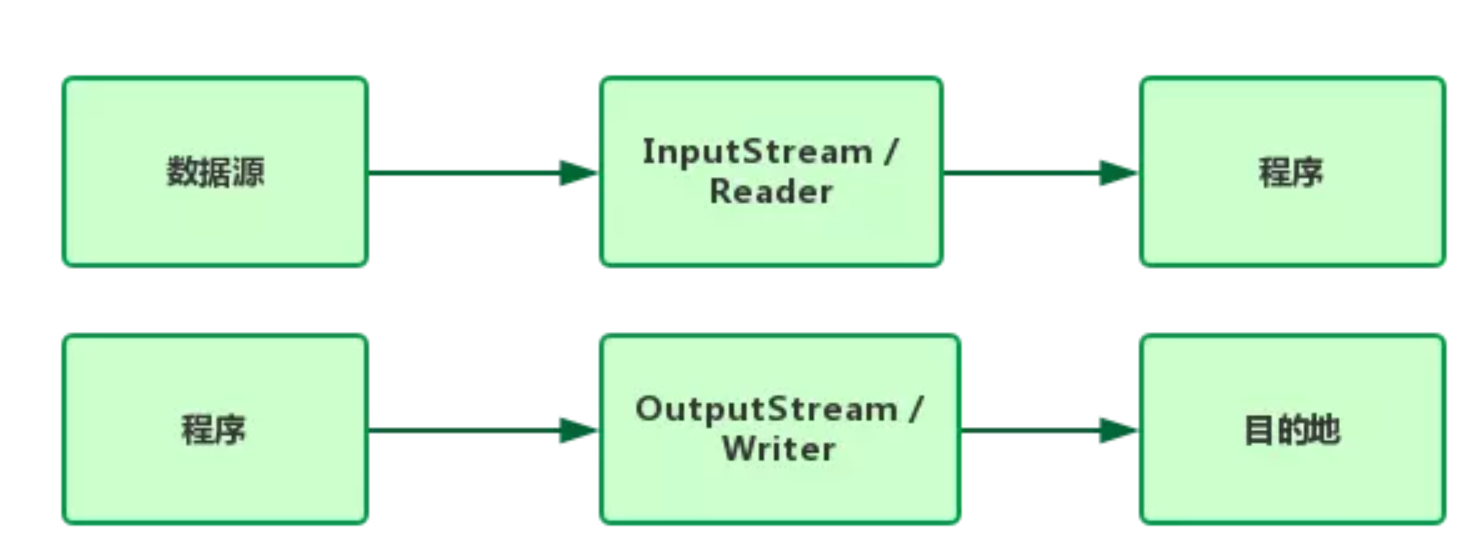


## 流

IO流是Java IO的核心概念。流是一个概念上的无休止的数据流。你既可以从一个流读取数据也可以写入到一个流。流用来连接数据源与目标地。Java IO流基于字节流和字符流。

## InputStream, OutputStream, Reader and Writer

一个程序从数据源读取数据的时候，需要 InputStream或者Reader。程序写数据的时候，就需要OutputStream或Writer。具体可以参见下面图例：



InputStream和Reader连接到数据来源。OutputStream和Writer连接到目的地

## Java IO的目的和特性

Java IO中有很多InputStream, OutputStream, Reader和Writer的子类，这些子类会用来处理不同的需求。这就是为什么有这么多类的原因。具体的用法主要有以下一些场景：

* 文件
* 网络流
* 内存缓冲区
* 管道
* 缓冲
* 过滤
* 解析
* 读写文本
* 读写原始类型的数据（long等等）
* 读写对象

上面的这些可以让你在阅读Java IO相关类的时候清晰一些。更能理解这些类的目的都是什么。

## Java IO相关类概览

在讨论源，目的地， input，output和各式各样的Java IO类的时候，下面的表以input和output做分类，基本列出了大部分的Java IO相关的类，有字符流，也有字节流，以及更多特殊用途的如buffering,，parsing等等：

|  | **字节输入流** | **Output** | **字符输入流** | **Output** |
| --- | --- | --- | --- | --- |
| 基础类 | InputStream | OutputStream | Reader/InputStreamReader | Writer/OutputStreamWriter |
| 数组 | ByteArrayInputStream | ByteArrayOutputStream | CharArrayReader | CharArrayWriter |
| 文件 | FileInputStream/RandomAccessFile | FileOutputStream/RandomAccessFile | FileReader | FileWriter |
| 管道 | PipedInputStream | PipedOutputStream | PipedReader | PipedWriter |
| Buffer | BufferedInputStream | BufferedOutputStream | BufferedReader | BufferedWriter |
| Filter | FilterInputStream | FilterOutputStream | FilterReader | FilterWriter |
| 解析 | PushbackInputStream/StreamTokenizer |  | PushbackReader/LineNumberReader |  |
| String |  |  | StringReader | StringWriter |
| Data | DataInputStream | DataOutputStream |  |  |
| Data - Formatted |  | PrintStream |  | PrintWriter |
| Objects | ObjectInputStream | ObjectOutputStream |  |  |
| 实用工具 | SequenceInputStream |  |  |  |

# Java IO: Files

在Java应用中，文件是常见的数据源。所以这篇文章会给你一个用java处理文件的简短介绍。这里虽然不会去详细阐述每一个知识点，但是也会给你足够的知识去决定如何处理文件。后面会有一些独立的章节，更加详细的去描述这些方法或类，以及一些例子等等。

## 根据Java IO读取文件

如果你想从一端读取一个文件到另一端，你可以用FileInputStream或者FileReader，这取决于你是想把文件读取成字节还是字符。这两个类每次读取一个字节或者一个字符来处理整个文件，或者将字节或字符放入到字节数组或字符数组中。你没有必要读取整个文件，只需要有序的读取文件中的字符或字节就可以。

如果你需要读取文件中随机位置的部分，你可以使用RandomAccessFile。

## 根据Java IO写文件

如果你想从一端读取一个文件到另一端，你可以用FileOutputStream或者FileWriter，这取决于你是想根据字节还是字符来写。这两个类每次写入一个字节或者一个字符来写完整个文件，或者将字节或字符放入到字节数组或字符数组中来写。数据会按顺序存储到文件当中。

如果你需要在文件中随机的位置来写，你可以使用RandomAccessFile。

## 根据Java IO随机访问文件

上面的内容已经说到，你可以利用RandomAccessFile来随机访问文件。

随机的意思并是不真正意义上的随机访问文件地址，它紧紧是意味着你可以根据你的需要来访问文件的某个位置。没有一个强制的顺序让你访问文件。这就让处理已经存在的文件的某部分称为了可能，去增加内容，从中删除内容，当然了也包括去读取任何内容。

## 获取文件和目录信息

有时候比起文件内容，你更想去获取文件信息。比如说你想获取文件或者目录的大小，以及文件的其他属性。再比如获取某目录下的所有文件列表。获取目录和文件的信息，都可以通过Java IO的类来解决。

# Java IO: Pipes

Java IO中,管道用来提供同一个jvm中的两个线程间的通信。所以管道也可以是数据的源或目的地。在不同的jvm（不同的进程）下，你不可以用管道去连接线程。Java的管道概念，不同于linux或unix的管道，linux或unix中，管道可以用来连接两个运行在不同地址空间的进程。（译者注：linux下的管道符为“ | ”，例如：ps -ef | grep java）。Java中，通讯部分必须在同一个进程下，可以使不同线程。

## 利用Java IO创建管道

用Java IO创建管道，已经有线程的类可以使用：PipedOutputStream和PipedInputStream。PipedInputStream可以用来连接PipedOutputStream。一个线程把数据写入到PipedOutputStream，可以被另一个线程创建的已经连接的PipedInputStream读取。

## Java IO管道示例

这里提供一个简单的例子来演示如何去连接一个PipedInputStream到一个PipedOutputStream：

除了利用构造方法外,你也可以用connect()方法来连接两个管道流。PipedInputStream和PipedOutputStream都有connect()方法。

## 管道和线程

记住这一点，当使用两个已连接的管道流时，流和线程应该是一一对应的关系。read()和write()方法是阻塞的方法，这意味着，如果你用同一个线程既读又写，那么就会造成这个线程的死锁。

## 管道的替代方案

在同一jvm中，线程间通讯有很多其他的办法。实际上线程间的交互，相比利用原始字节数据，更多的还是利用对象进行交互。当然了，如果你想利用原始数据进行线程间的交互，Java IO的管道流是一个不错的选择。