

- [首页](#)
- [所有文章](#)
- [资讯](#)
- [Web](#)
- [架构](#)
- [基础技术](#)
- [书籍](#)
- [教程](#)
- [Java小组](#)
- [工具资源](#)

- 导航条 - ▼


Java NIO系列教程（12）：Java NIO与IO

2016/04/08 | 分类：[教程](#) | [0 条评论](#) | 标签：[io](#), [Java NIO](#)

分享到：

²⁴ 译文出处：[郭董](#) 原文出处：[Jakob Jenkov](#)

当学习了Java NIO和IO的API后，一个问题马上涌入脑海：

我应该何时使用IO，何时使用NIO呢？在本文中，我会尽量清晰地解析Java NIO和IO的差异、它们的使用场景，以及它们如何影响您的代码设计。

Java NIO和IO的主要区别

下表总结了Java NIO和IO之间的主要差别，我会更详细地描述表中每部分的差异。

IO	NIO
面向流	面向缓冲
阻塞IO	非阻塞IO
无	选择器

面向流与面向缓冲

Java NIO和IO之间第一个最大的区别是，IO是面向流的，NIO是面向缓冲区的。Java IO面向流意味着每次从流中读一个或多个字节，直至读取所有字节，它们没有被缓存在任何地方。此外，它不能前后移动流中的数据。如果需要前后移动从流中读取的数据，需要先将它缓存到一个缓冲区。Java NIO的缓冲导向方法略有不同。数据读取到一个它稍后处理的缓冲区，需要时可在缓冲区中前后移动。这就增加了处理过程中的灵活性。但是，还需要检查是否该缓冲区中包含所有您需要处理的数据。而且，需确保当更多的数据读入缓冲区时，不要覆盖缓冲区里尚未处理的数据。

阻塞与非阻塞IO

Java IO的各种流是阻塞的。这意味着，当一个线程调用read() 或 write()时，该线程被阻塞，直到有一些数据被读取，或数据完全写入。该线程在此期间不能再干任何事情了。Java NIO的非阻塞模式，使一个线程从某通道发送请求读取数据，但是它仅能得到目前可用的数据，如果目前没有数据可用时，就什么也不会获取。而不是保持线程阻塞，所以直至数据变的可以读取之前，该线程可以继续做其他的事情。非阻塞写也是如此。一个线程请求写入一些数据到某通道，但不需要等待它完全写入，这个线程同时可以去做别的事情。线程通常将非阻塞IO的空闲时间用于在其它通道上执行IO操作，所以一个单独的线程现在可以管理多个输入和输出通道（channel）。

选择器（Selectors）

Java NIO的选择器允许一个单独的线程来监视多个输入通道，你可以注册多个通道使用一个选择器，然后使用一个单独的线程来“选择”通道：这些通道里已经有可以处理的输入，或者选择已准备写入的通道。这种选择机制，使得一个单独的线程很容易来管理多个通道。

NIO和IO如何影响应用程序的设计

无论您选择IO或NIO工具箱，可能会影响您应用程序设计的以下几个方面：

1. 对NIO或IO类的API调用。
2. 数据处理。
3. 用来处理数据的线程数。



API调用

当然，使用NIO的API调用时看起来与使用IO时有所不同，但这并不意外，因为并不是仅从一个InputStream逐字节读取，而是数据必须先读入缓冲区再处理。

数据处理

使用纯粹的NIO设计相较IO设计，数据处理也受到影响。

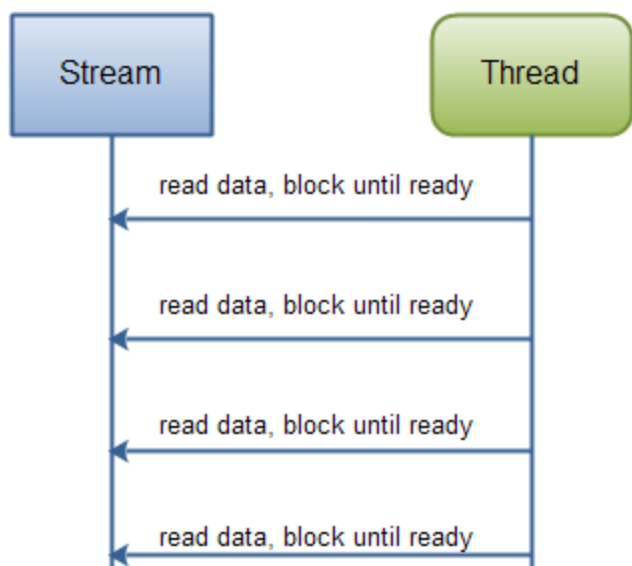
在IO设计中，我们从InputStream或 Reader逐字节读取数据。假设你正在处理一基于行的文本数据流，例如：

```
1 Name: Anna
2 Age: 25
3 Email: anna@mailserver.com
4 Phone: 1234567890
```

该文本行的流可以这样处理：

```
1 InputStream input = ... ; // get the InputStream from the client socket
2
3 BufferedReader reader = new BufferedReader(new InputStreamReader(input));
4
5 String nameLine    = reader.readLine();
6 String ageLine     = reader.readLine();
7 String emailLine   = reader.readLine();
8 String phoneLine   = reader.readLine();
```

请注意处理状态由程序执行多久决定。换句话说，一旦`reader.readLine()`方法返回，你就知道肯定文本行就已读完，`readline()`阻塞直到整行读完，这就是原因。你也知道此行包含名称；同样，第二个`readline()`调用返回的时候，你知道这行包含年龄等。正如你可以看到，该处理程序仅在有新数据读入时运行，并知道每步的数据是什么。一旦正在运行的线程已处理过读入的某些数据，该线程不会再回退数据（大多如此）。下图也说明了这条原则：



Java IO: 从一个阻塞的流中读数据

而一个NIO的实现会有所不同，下面是一个简单的例子：

```
1 | ByteBuffer buffer = ByteBuffer.allocate(48);
2 | int bytesRead = inChannel.read(buffer);
```



注意第二行，从通道读取字节到`ByteBuffer`。当这个方法调用返回时，你不知道你所需的所有数据是否在缓冲区内。你所知道的是，该缓冲区包含一些字节，这使得处理有点困难。

假设第一次 `read(buffer)` 调用后，读入缓冲区的数据只有半行，例如，“Name:An”，你能处理数据吗？显然不能，需要等待，直到整行数据读入缓存，在此之前，对数据的任何处理毫无意义。

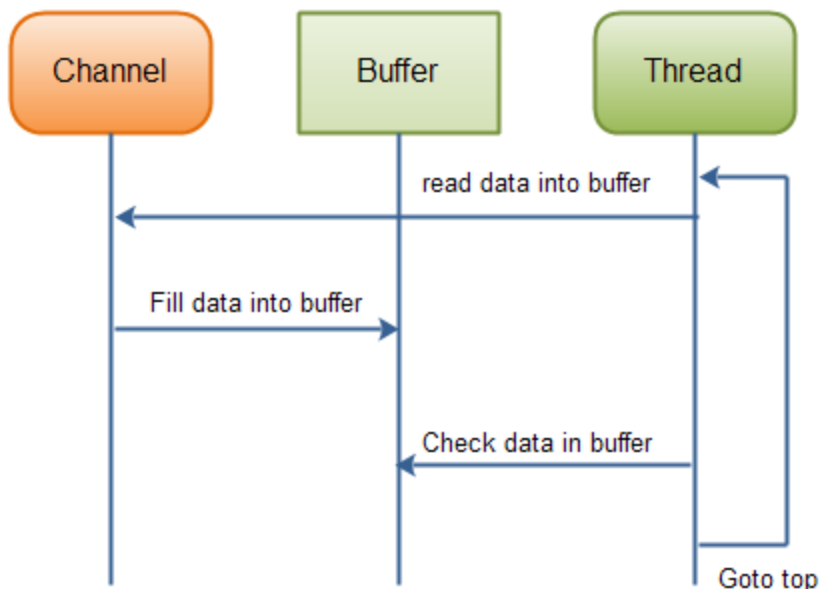
所以，你怎么知道是否该缓冲区包含足够的数据可以处理呢？好了，你不知道。发现的方法只能查看缓冲区中的数据。其结果是，在你知道所有数据都在缓冲区里之前，你必须检查几次缓冲区的数据。这不仅效率低下，而且可以使程序设计方案杂乱不堪。例如：

```
1 | ByteBuffer buffer = ByteBuffer.allocate(48);
2 |
3 | int bytesRead = inChannel.read(buffer);
4 |
5 | while(! bufferFull(bytesRead) ) {
6 |
7 |     bytesRead = inChannel.read(buffer);
8 |
9 | }
```

`bufferFull()`方法必须跟踪有多少数据读入缓冲区，并返回真或假，这取决于缓冲区是否已满。换句话说，如果缓冲区准备好被处理，那么表示缓冲区满了。

`bufferFull()`方法扫描缓冲区，但必须保持在`bufferFull()`方法被调用之前状态相同。如果没有，下一个读入缓冲区的数据可能无法读到正确的位置。这是不可能的，但却是需要注意的又一问题。

如果缓冲区已满，它可以被处理。如果它不满，并且在你的实际案例中有意义，你或许能处理其中的部分数据。但是许多情况下并非如此。下图展示了“缓冲区数据循环就绪”：

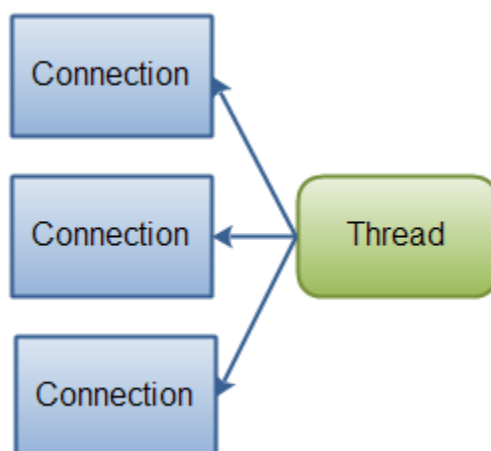


Java NIO:从一个通道里读数据，直到所有的数据都读到缓冲区里。

总结

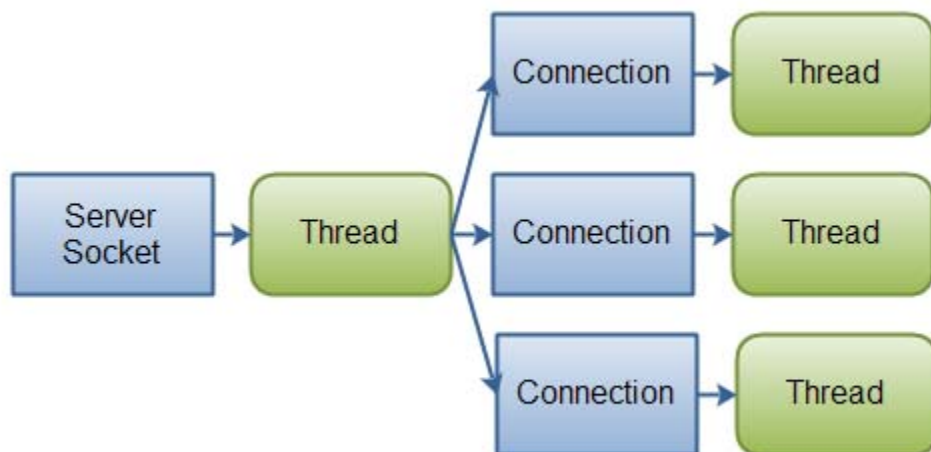
NIO可让您只使用一个（或几个）单线程管理多个通道（网络连接或文件），但付出的代价是解析数据可能会比从一个阻塞流中读取数据更复杂。 

如果需要管理同时打开的成千上万个连接，这些连接每次只是发送少量的数据，例如聊天服务器，实现NIO的服务器可能是一个优势。同样，如果你需要维持许多打开的连接到其他计算机上，如P2P网络中，使用一个单独的线程来管理你所有出站连接，可能是一个优势。一个线程多个连接的设计方案如下图所示：



Java NIO: 单线程管理多个连接

如果你有少量的连接使用非常高的带宽，一次发送大量的数据，也许典型的IO服务器实现可能非常契合。下图说明了一个典型的IO服务器设计：



Java IO: 一个典型的IO服务器设计- 一个连接通过一个线程处理.

本系列：

- [Java NIO系列教程（1）：Java NIO 概述](#)
- [Java NIO系列教程（2）：Channel](#)
- [Java NIO系列教程（3）：Buffer](#)
- [Java NIO系列教程（4）：Scatter/Gather](#)
- [Java NIO系列教程（5）：通道之间的数据传输](#)
- [Java NIO系列教程（6）：Selector](#)
- [Java NIO系列教程（7）：FileChannel](#)
- [Java NIO系列教程（8）：SocketChannel](#)
- [Java NIO系列教程（9）：ServerSocketChannel](#)
- [Java NIO系列教程（10）：Java NIO DatagramChannel](#)
- [Java NIO系列教程（11）：Pipe](#)
- [Java NIO系列教程（12）：Java NIO与IO](#)

24



相关文章

- [Java 标准 I/O 流编程一览笔录](#)
- [理解Java中字符流与字节流的区别](#)
- [Java I/O 总结](#)
- [也谈IO模型](#)
- [Java 编程要点之 I/O 流详解](#)
- [Java I/O 模型的演进](#)
- [【Java TCP/IP Socket】Java NIO Socket VS 标准IO Socket](#)

- [java中的IO整理](#)
- [攻破JAVA NIO技术壁垒](#)
- [Java NIO系列教程（11）：Pipe](#)

发表评论

Comment form

Name*

姓名

邮箱*

请填写邮箱

网站 (请以 http://开头)

请填写网站地址

评论内容*

请填写评论内容



(*) 表示必填项

[提交评论](#)

还没有评论。

[« Java NIO系列教程（11）：Pipe
WebService性能测试 »](#)

Search for:

Search

Search



- [本周热门文章](#)
- [本月热门](#)
- [热门标签](#)

0 [记一次集群内无可用 http 服务问题...](#)

1 [Java 技术之垃圾回收机制](#)

2 [公司编程竞赛之最长路径问题](#)

3 [Java 中的十个"单行代码编程" \(O...](#)

4 [Java 中 9 个处理 Exception ...](#)

5 [HttpClient 以及 Json 传递的...](#)

6 [浅析 Spring 中的事件驱动机制](#)

7 [浅析分布式下的事件驱动机制 \(PubS...](#)

8 [探索各种随机函数 \(Java 环境...](#)

9 [Java 守护线程概述](#)



最新评论



Re: [攻破JAVA NIO技术壁垒](#)

Hi, 请到伯乐在线的小组发帖提问, 支持微信登录。链接是: <http://group.jobbole....> 唐尤华



Re: [攻破JAVA NIO技术壁垒](#)

TCP服务端的NIO写法 服务端怎么发送呢。原谅小白 菜鸟



Re: [关于 Java 中的 double check ...](#)

volatile 可以避免指令重排啊。所以double check还是可以用的。 hipilee



Re: [Spring4 + Spring MVC + M...](#)

Hi, 请到伯乐在线的小组发帖提问, 支持微信登录。链接是: <http://group.jobbole....> 唐尤华



Re: [Spring4 + Spring MVC + M...](#)

我的一直不太明白，spring的bean容器和springmvc的bean容器之间的关系。 hw_绝影



Re: [Spirng+SpringMVC+Maven+Myba...](#)

很好，按照步骤，已经成功。 莫凡



Re: [Spring中@Transactional事务...](#)

声明式事务可以用aop来实现,分别是jdk代理和cglib代理,基于接口和普通类.在同一个类中一个方... chengjiliang



Re: [关于 Java 中的 double check ...](#)

在JDK1.5之后，用volatile关键字修饰_INSTANCE属性 就能避免因指令重排导致的对象... Byron

关于ImportNew

ImportNew 专注于 Java 技术分享。于2012年11月11日 11:11正式上线。是的，这是一个很特别的时刻：)

ImportNew 由两个 Java 关键字 import 和 new 组成，意指：Java 开发者学习新知识的网站。import 可认为是学习和吸收，new 则可认为是新知识、新技术圈子和新朋友.....



联系我们

Email : ImportNew.com@gmail.com

新浪微博 : [@ImportNew](#)

推荐微信号



ImportNew



安卓应用频道



Linux爱好者

反馈建议 : ImportNew.com@gmail.com

广告与商务合作QQ : 2302462408

推荐关注

[小组](#) – 好的话题、有启发的回复、值得信赖的圈子

[头条](#) – 写了文章？看干货？去头条！

[相亲](#) – 为IT单身男女服务的征婚传播平台

[资源](#) – 优秀的工具资源导航

[翻译](#) – 活跃 & 专业的翻译小组
[博客](#) – 国内外的精选博客文章
[设计](#) – UI,网页, 交互和用户体验
[前端](#) – JavaScript, HTML5, CSS
[安卓](#) – 专注Android技术分享
[iOS](#) – 专注iOS技术分享
[Java](#) – 专注Java技术分享
[Python](#) – 专注Python技术分享

© 2017 ImportNew

