

# ImportNew

- [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（6）：Selector

2016/04/02 | 分类：[教程](#) | [1 条评论](#) | 标签：[Java NIO](#), [selector](#)

分享到：

<sup>21</sup> 译文出处：[浪迹v](#) 原文出处：[Jakob Jenkov](#)

Selector（选择器）是Java NIO中能够检测一到多个NIO通道，并能够知晓通道是否为诸如读写事件做好准备的组件。这样，一个单独的线程可以管理多个channel，从而管理多个网络连接。

下面是本文所涉及到的主题列表：

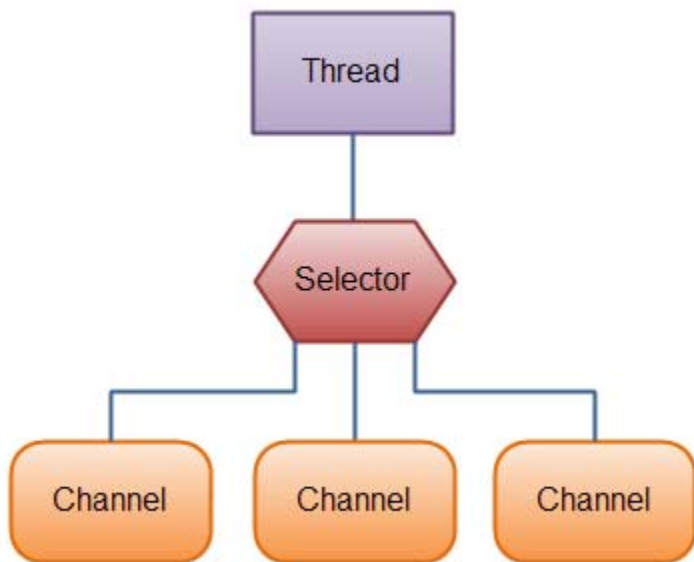
1. 为什么使用Selector?
2. Selector的创建
3. 向Selector注册通道
4. SelectionKey
5. 通过Selector选择通道
6. wakeUp()
7. close()
8. 完整的示例

### 为什么使用Selector?

仅用单个线程来处理多个Channels的好处是，只需要更少的线程来处理通道。事实上，可以只用一个线程处理所有的通道。对于操作系统来说，线程之间上下文切换的开销很大，而且每个线程都要占用系统的一些资源（如内存）。因此，使用的线程越少越好。

但是，需要记住，现代的操作系统和CPU在多任务方面表现的越来越好，所以多线程的开销随着时间的推移，变得越来越小了。实际上，如果一个CPU有多个内核，不使用多任务可能是在浪费CPU能力。不管怎么说，关于那种设计的讨论应该放在另一篇不同的文章中。在这里，只要知道使用Selector能够处理多个通道就足够了。

下面是单线程使用一个Selector处理3个channel的示例图：



## Selector的创建

通过调用Selector.open()方法创建一个Selector，如下：

```
1 | Selector selector = Selector.open();
```

## 向Selector注册通道

为了将Channel和Selector配合使用，必须将channel注册到selector上。通过SelectableChannel.register()方法来实现，如下：

```
1 | channel.configureBlocking(false);
2 | SelectionKey key = channel.register(selector,
3 |     SelectionKey.OP_READ);
```



与Selector一起使用时，Channel必须处于非阻塞模式下。这意味着不能将FileChannel与Selector一起使用，因为FileChannel不能切换到非阻塞模式。而套接字通道都可以。

注意register()方法的第二个参数。这是一个“interest集合”，意思是在通过Selector监听Channel时对什么事件感兴趣。可以监听四种不同类型的事件：

1. Connect
2. Accept
3. Read
4. Write

通道触发了一个事件意思是该事件已经就绪。所以，某个channel成功连接到另一个服务器称为“连接就绪”。一个server socket channel准备好接收新进入的连接称为“接收就绪”。一个有数据可读的通道可以说是“读就绪”。等待写数据的通道可以说是“写就绪”。

这四种事件用SelectionKey的四个常量来表示：

1. SelectionKey.OP\_CONNECT
2. SelectionKey.OP\_ACCEPT
3. SelectionKey.OP\_READ
4. SelectionKey.OP\_WRITE

如果你对不止一种事件感兴趣，那么可以用“位或”操作符将常量连接起来，如下：

```
1 | int interestSet = SelectionKey.OP_READ | SelectionKey.OP_WRITE;
```

在下面还会继续提到interest集合。

## SelectionKey

在上一小节中，当向Selector注册Channel时，register()方法会返回一个SelectionKey对象。这个对象包含了一些你感兴趣的属性：

- interest集合
- ready集合
- Channel
- Selector
- 附加的对象（可选）

下面我会描述这些属性。

## interest集合

就像向Selector注册通道一节中所描述的，interest集合是你所选择的感兴趣的事件集合。可以通过SelectionKey读写interest集合，像这样：

```
1 | int interestSet = selectionKey.interestOps();
2
3 | boolean isInterestedInAccept = (interestSet & SelectionKey.OP_ACCEPT) == SelectionKey.OP_ACCEPT;
4 | boolean isInterestedInConnect = interestSet & SelectionKey.OP_CONNECT;
5 | boolean isInterestedInRead    = interestSet & SelectionKey.OP_READ;
6 | boolean isInterestedInWrite   = interestSet & SelectionKey.OP_WRITE;
```



可以看到，用“位与”操作interest集合和给定的SelectionKey常量，可以确定某个确定的事件是否在interest集合中。

## ready集合

ready集合是通道已经准备就绪的操作的集合。在一次选择(Selection)之后，你会首先访问这个ready set。Selection将在下一小节进行解释。可以这样访问ready集合：

```
1 | int readySet = selectionKey.readyOps();
```

可以用像检测interest集合那样的方法，来检测channel中什么事件或操作已经就绪。但是，也可以使用以下四个方法，它们都会返回一个布尔类型：

```
1 | selectionKey.isAcceptable();
2 | selectionKey.isConnectable();
3 | selectionKey.isReadable();
4 | selectionKey.isWritable();
```

## Channel + Selector

从SelectionKey访问Channel和Selector很简单。如下：

```
1 | Channel channel = selectionKey.channel();
2 | Selector selector = selectionKey.selector();
```

## 附加的对象

可以将一个对象或者更多信息附着到SelectionKey上，这样就能方便的识别某个给定的通道。例如，可以附加 与通道一起使用的Buffer，或是包含聚集数据的某个对象。使用方法如下：

```
1 | selectionKey.attach(theObject);
2 | Object attachedObj = selectionKey.attachment();
```

还可以在用register()方法向Selector注册Channel的时候附加对象。如：

```
1 | SelectionKey key = channel.register(selector, SelectionKey.OP_READ, theObject);
```

## 通过Selector选择通道


一旦向Selector注册了一或多个通道，就可以调用几个重载的select()方法。这些方法返回你所感兴趣的事件（如连接、接受、读或写）已经准备就绪的那些通道。换句话说，如果你对“读就绪”的通道感兴趣，select()方法会返回读事件已经就绪的那些通道。

下面是select()方法：

- int select()
- int select(long timeout)
- int selectNow()

select()阻塞到至少有一个通道在你注册的事件上就绪了。

select(long timeout)和select()一样，除了最长会阻塞timeout毫秒(参数)。

selectNow()不会阻塞，不管什么通道就绪都立刻返回（译者注：此方法执行非阻塞的选择操作。如果自从前一次选择操作后，没有通道变成可选择的，此方法直接返回零。）。

select()方法返回的int值表示有多少通道已经就绪。亦即，自上次调用select()方法后有多少通道变成就绪状态。如果调用select()方法，因为有一个通道变成就绪状态，返回了1，若再次调用select()方法，如果另一个通道就绪了，它会再次返回1。如果对第一个就绪的channel没有做任何操作，现在就有两个就绪的通道，但在每次select()方法调用之间，只有一个通道就绪了。

## selectedKeys()

一旦调用了select()方法，并且返回值表明有一个或多个通道就绪了，然后通过调用selector的selectedKeys()方法，访问“已选择键集（selected key set）”中的就绪通道。如下所示：

```
1 | Set selectedKeys = selector.selectedKeys();
```

当像Selector注册Channel时，Channel.register()方法会返回一个SelectionKey 对象。这个对象代表了注册到该Selector的通道。可以通过SelectionKey的selectedKeySet()方法访问这些对象。

可以遍历这个已选择的键集合来访问就绪的通道。如下：

```
1 | Set selectedKeys = selector.selectedKeys();
2 | Iterator keyIterator = selectedKeys.iterator();
3 | while(keyIterator.hasNext()) {
4 |     SelectionKey key = keyIterator.next();
5 |     if(key.isAcceptable()) {
6 |         // a connection was accepted by a ServerSocketChannel.
7 |     } else if (key.isConnectable()) {
8 |         // a connection was established with a remote server.
9 |     } else if (key.isReadable()) {
```

```
10         // a channel is ready for reading
11     } else if (key.isWritable()) {
12         // a channel is ready for writing
13     }
14     keyIterator.remove();
15 }
```

这个循环遍历已选择键集中的每个键，并检测各个键所对应的通道的就绪事件。

注意每次迭代末尾的keyIterator.remove()调用。Selector不会自己从已选择键集中移除SelectionKey实例。必须在处理完通道时自己移除。下次该通道变成就绪时，Selector会再次将其放入已选择键集中。

SelectionKey.channel()方法返回的通道需要转型成你要处理的类型，如ServerSocketChannel或SocketChannel等。

## wakeup()

某个线程调用select()方法后阻塞了，即使没有通道已经就绪，也有办法让其从select()方法返回。只要让其它线程在第一个线程调用select()方法的那个对象上调用Selector.wakeup()方法即可。阻塞在select()方法上的线程会立马返回。

如果有其它线程调用了wakeup()方法，但当前没有线程阻塞在select()方法上，下个调用select()方法的线程会立即“醒来（wake up）”。

## close()

用完Selector后调用其close()方法会关闭该Selector，且使注册到该Selector上的所有SelectionKey实例无效。通道本身并不会关闭。



## 完整的示例

这里有一个完整的示例，打开一个Selector，注册一个通道注册到这个Selector上(通道的初始化过程略去),然后持续监控这个Selector的四种事件（接受，连接，读，写）是否就绪。

```
1  Selector selector = Selector.open();
2  channel.configureBlocking(false);
3  SelectionKey key = channel.register(selector, SelectionKey.OP_READ);
4  while(true) {
5      int readyChannels = selector.select();
6      if(readyChannels == 0) continue;
7      Set selectedKeys = selector.selectedKeys();
8      Iterator keyIterator = selectedKeys.iterator();
9      while(keyIterator.hasNext()) {
10         SelectionKey key = keyIterator.next();
11         if(key.isAcceptable()) {
12             // a connection was accepted by a ServerSocketChannel.
13         } else if (key.isConnectable()) {
14             // a connection was established with a remote server.
15         } else if (key.isReadable()) {
16             // a channel is ready for reading
17         } else if (key.isWritable()) {
18             // a channel is ready for writing
19         }
20         keyIterator.remove();
21     }
22 }
```

## 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](#)

21



## 相关文章

- [Java NIO系列教程（7）：FileChannel](#)
- [Selector 实现原理](#)
- [epoll 浅析以及 nio 中的 Selector](#)
- [攻破JAVA NIO技术壁垒](#)
- [Java NIO系列教程（12）：Java NIO与IO](#)
- [Java NIO系列教程（11）：Pipe](#)
- [Java NIO系列教程（10）：Java NIO DatagramChannel](#)
- [Java NIO系列教程（9）：ServerSocketChannel](#)
- [Java NIO系列教程（8）：SocketChannel](#)
- [Java NIO系列教程（5）：通道之间的数据传输](#)



## 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](#)

## 1 条评论

1. [加速奔跑的蜗牛](#) 说道：

[2017/08/08 下午 5:30](#)

楼主最后一个完整代码，key怎么定义了两次？有错吧



0 0

[回复](#)

[« Java NIO系列教程（5）：通道之间的数据传输](#)

[Java NIO系列教程（7）：FileChannel »](#)

Search for:



- [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: [Spring+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](mailto: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](mailto: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

