# 选择器基础

通道处于就绪状态后，就可以在缓冲区之间传送数据。可以采用非阻塞模式来检查通道是否就绪，但非阻塞模式还会做别的任务，当有多个通道同时存在时，很难将检查通道是否就绪与其他任务玻璃开来，或者说是这样做很复杂，即使完成了这样的功能，但每检查一次通道的就绪状态，就至少有一次系统调用，代价十分昂贵。当您轮询每个通道的就绪状态时，刚被检查的一个处于未就绪状态的通道，突然处于就绪状态，在下一次轮询之前是不会被察觉的。操作系统拥有这种检查就绪状态并通知就绪的能力，因此要充分利用操作系统提供的服务。在JAVA中，Selector类提供了这种抽象，拥有询问通道是否已经准备好执行每个I/O操作的能力，所以可以利用选择器来很好地解决以上问题。

**流程结构图：**



**选择器（Selector）**

选择器类管理着一个被注册的通道集合的信息和他们的就绪状态。通道是和选择器一起被注册的，并且使用选择器来更新通道的 就绪状态。

可选择通道（SelectableChannel）

这个抽象类提供了通道的可选择所需要的公共方法，FileChannel对象不是可选择的，因为他们没有继承SelectableChannel。所有socket通道都是可选择的。包括从管道（Pipe）对象中获得的通道。SelectableChannel可以被注册到Selector对象上，一个通道可以被注册到多个选择器上，但对每个选择器而言只能被注册一次。

选择键（SelectionKey）

选择键封装了通道与选择器的注册关系。选择键对象被SelectableChannel.register返回并提供一个表示这种注册关系的标记。通道在被注册到一个选择器上之前，必须先设置为非阻塞模式（通过调用configureBloking(false)）。

调用可选择通道的register()方法会将它注册到一个选择器上。如果视图注册一个处于阻塞状态的通道，register()将抛出未检查的IllegalBlockingModeException异常。此外，通道一旦被注册，就不能回到阻塞状态。试图这么做的话，将在调用configureBlocking()方法时将抛出IllegalBlockingModeException异常。并且，试图注册一个已经关闭的SelectableChannel实例的话，也将抛出ClosedChannelException异常。

键的interest（感兴趣的操作）集合和ready（已经准备好的操作）集合是和特定的通道相关的。每个通道的实现，将定义它自己的选择键类。在register()方法中可以构造它并将它传递给所提供的选择器对象。

如果我们要使用非阻塞I/O编写服务器处理程序，大致的步骤如下：

1、向Selector对象注册感兴趣的事件。

2、从Selector中获取感兴趣的事件。

3、根据不同的事件进行相应的处理。

# 建立选择器

通过以下代码我们可以建立监控三个Socket通道的选择器：

Selector selector = Selector.open();

channel1.register(selector,SelectionKey.OP\_READ);

channel2.register(selector,SelectionKey.OP\_WRITE);

channel3.register(selector,SelectionKey.OP\_READ | SelectionKey.OP\_WRITE);

// Wait up to 10 seconds for a channel to become ready

readyCount = selector.select(10000);

select方法时阻塞方法，直到过了十秒或者至少有一个通道的I/O操作准备好。

# 使用选择器

如下给出了一些选择器方法：

|  |  |
| --- | --- |
| abstract void close() | 关闭此选择器 |
| abstract Boolean isOpen() | 判断选择器是否已打开 |
| static Selector open() | 打开一个选择器 |
| abstract SelectorProvider provider() | 返回创建此通道的提供者 |
| abstract int select() | 返回一组键的个数，其相应的通道已为I/O操作准备就绪 |
| abstract int select(long timeout) | 同上，指定了阻塞时间 |
| abstract int selectNow() | select()方法的非阻塞形式。不等于select(0)（无限期阻塞） |
| abstract Set keys() | 返回此选择器的键集 |
| abstract Set selectedKeys() | 返回此选择器上相应的通道I/O操作准备就绪的选择键集 |
| abstract Selector wakeup() | 使尚未返回的第一个选择操作立即返回 |

关于选择键的方法：

|  |  |
| --- | --- |
| Object attach(Object ob) | 将给定的对象附加到此键 |
| Object attachment() | 获取当前的附加对象 |
| abstract void cancel() | 请求取消此键的通道到其选择器的注册 |
| abstract Boolean isValid() | 告知此键是否有效 |
| abstract SelectableChannel channel() | 返回与此键相关的通道 |
| abstract Selector selector() | 返回此选择器创建的键 |
| abstract int interestOps() | 获取此键的interest集合，是通道被注册时传进来的值 |
| abstract SelectionKey interestOps(int ops) | 将此键的interest集合设置为给定值 |
| abstract int readyOps() | 获取此键的ready操作集合，为interest的子集 |
| boolean isAcceptable() | 测试此键的通道是否已经准备好接受新的套接字连接 |
| boolean isConnectable() | 测试此键的通道是否已经完成其套接字连接操作 |
| boolean isReadable() | 测试此键的通道是否已经准备好进行读取 |
| boolean isWritable() | 测试此键的通道是否已经准备好进行写入 |

在SelectionKey中，用静态常量定义了四种I/O操作：OP\_READ 1、 OP\_WRITE 4、OP\_CONNECT 8、OP\_ACCEPT 16，这4个值任何2、3、4个相加结果都不相同，因此可以用validOps()方法返回值确定SelectableChannel支持的操作。

当通道关闭时，所有相关的键会自动取消。当选择器关闭时，所有被注册到该选择的通道都将被注销，并且相关的键将立即被无效化（取消）。一旦键被无效化，调用它的与选择器相关的方法就将抛出CancelledKeyException。

前面已经列出了关于选择方法列表，下面介绍基本用法。

通过keys()方法已注册键的集合，集合可能是空的。这个已注册的键的集合不是可以直接修改的，试图这么做的话将java.lang.UnsupportedOperationException。

已选择的键的集合(Selected key set)是已注册的键的集合的子集，这个集合的每个成员都是相关的通道被选择判断为已经准备好的。这个集合通过selectedKeys()方法返回（可能是空的）。

已取消的键的集合（Cancelled key set）是已注册的键的集合的子集。这个集合包含了cancel()方法被调用过的键（这个键已经被无效化），但它们还没有被注销，这个集合是选择器对象的私有成员，因而无法直接访问。

以下三种方式可以唤醒select()方法中睡眠的现成：

a、调用Selector对象的wakeup()方法将使得选择上的第一个还没有返回的选择操作立即返回。如果当前没有在进行中的选择，那么下一次对select()方法的一种形式的调用将立即返回。后续的选择操作将正常进行。在选择操作之间多次调用wakeup()方法与调用它一次没有什么不同。

b、如果选择器的close()方法被调用，那么任何一个在选择操作中阻塞的线程都将被唤醒，就像wakeup()方法被调用了一样。但是与选择器相关的通道将被注销，键将被取消。

c、如果睡眠中的线程interupt()方法被调用，它的返回状态被设置。如果被唤醒的线程之后将试图在通道上执行I/O操作，通道将立即关闭，然后线程将捕捉到一个异常。Selector对象将被捕捉InterruptedException异常并调用wakeup()方法。