Programación en Red: Canales y Selectores

Canales

Un **canal** representa una conexión abierta hacia una entidad como un dispositivo de hardware, un archivo, un socket de red o un componente del programa capaz de realizar una o más operaciones I/O distintas.

Java implementa la interfaz **Channel** del paquete **java.nio** para representar los canales. Se tienen únicamente dos métodos:

- ✓ void close( )
- ✓ boolean isOpen( )

Todo canal se encuentra abierto al ser creado, y debe ser cerrado haciendo uso del método close( ). Una vez cerrado, todo intento de invocar operaciones I/O arrojaran una **ClosedChannelException**.

Server Socket Channel

- Clase del paquete java.nio.channels

Un ServerSocketChannel es un canal seleccionable para sockets escuchando streams.

Para instanciar esta clase es necesario invocar a su método estático open( ), el cual retorna un objeto ServerSocketChannel.

```
ServerSocketChannel  serverChannel ;
serverChannel  =  ServerSocketChannel.open( );
```

Un canal de server-socket recién creado se encuentra abierto mas no vinculado aún. Cualquier intento para invocar al método accept( ) sobre un canal de server-socket arrojará un NotYetBoundException.

Socket Address

- Clase del paquete java.net

Un **SocketAddress** es una clase que representa una dirección IP de socket (IP Address + port ó hostname + port). Proporciona un objeto inmutable usado por los sockets para vincularse (binding), conectarse o como valores de retorno.

Para instanciar esta clase se puede cualquiera de los tres siguientes constructores:

- ✓ InetSocketAddress(InetAddress addr, int port)
- ✓ InetSocketAddress(int port)
- ✓ InetSocketAddress(String hostname, int port)

Vinculando Server Sockets Channels

Para poder aceptar conexiones a través de un canal de socket es necesario vincular el canal creado a una dirección de socket (dirección IP y puerto) haciendo uso de su método bind.

- ServerSocketChannel bind(SocketAddress endpoint)
  - El canal de retorno es simplemente una referencia a este mismo canal.
  - Si el canal ya se encuentra vinculado, arrojará un AlreadyBoundException.

Luego, se podrá invocar al método accept para aceptar conexiones entrantes.

- SocketChannel accept( )

Nota: aceptar una conexión a través de un canal de socket de servidor nos genera un canal de socket de cliente.

Generando Sockets

Se puede hacer uso de los canales de sockets para generar sockets asociados a dicho canal. Para ello, tanto los ServerSocketChannel como los SocketChannel cuentan con un método socket que retorna un socket asociado.

- abstract ServerSocket socket( )

Y en el caso de SocketChannel,

- abstract Socket socket( )

Si el canal se encontraba vinculado a alguna dirección, los socket retornados también lo estarán. Caso contrario se puede vincular un socket llamando al método bind del mismo socket.

Modo de Bloqueo de un Canal

Los ServerSocketChannel y SocketChannel son una subclase de un canal seleccionable (**AbstractSelectableChannel**) , por ende hereda todos los métodos de dicha clase abstracta y con ello el modo de bloqueo de un canal seleccionable.

Un canal en modo de bloqueo bloqueará toda operación I/O hasta que se complete (e.g. cuando se bloquea el hilo al llamar a una operación de lectura). Por defecto, todo canal creado se encuentra en modo de bloqueo.

Para modificar el modo de bloqueo de un canal seleccionable se puede hacer uso del siguiente método:

- final SelectableChannel configureBlocking(boolean block)
  - El argumento indica si el canal se encontrará en modo de bloque o no.
  - Se retorna una referencia a dicho canal seleccionable.
  - Puede arrojar una IOException o una ClosedChannelException.

Multiplexación e IO No Bloqueada

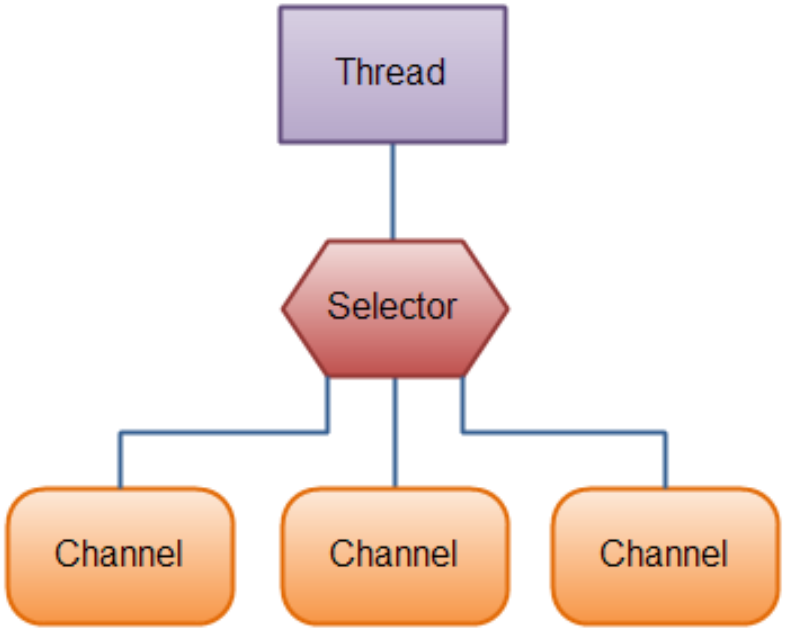
El uso de canales seleccionables permite implementar multiplexación además del modo no bloqueado para las operaciones IO.

Ejemplo ilustrativo:

> *Supongamos que se tiene un mesero atendiendo a una mesa en un restaurante. Si el administrador tuviera que asignar un mesero por mesa entonces el restaurante se iría fácilmente a la quiebra. En lugar de ello, un mesero atiene varias mesas (multiplexación) y no espera permanente en cada mesa a recibir la orden (IO no bloqueada).*

De forma análoga, veremos que haciendo uso de canales seleccionables y un **selector** podremos manejar varios canales en un solo hilo sin la necesidad de instanciar un hilo por cada socket abierto.

Haciendo uso de un selector es posible manejar múltiples canales a través de un solo hilo, con lo que se pueden asignar más hilos a otras tareas distintas.



Selector

Un **selector** es un componente de Java NIO capaz de examinar uno o más canales seleccionables NIO y determinar cual de ellos se encuentra listo para para leer o escribir.

Creando un selector

Un selector puede ser creado invocando el método estático open de la clase Selector.

```
try {
    Selector s = Selector.open( )
}
catch (IOException e) {
}
```

Todo selector se mantiene abierto desde su creación hasta que se llame al método close() de su instancia.

Selection Key

Un selection key es un token que representa la registración de un canal seleccionable (SelectableChannel) con un selector.

Métodos básicos:

- boolean isAcceptable( )
  - Verifica si el canal se encuentra listo para aceptar una nueva conexión de socket.
- boolean isConnectable( )
  - Verifica si el canal ha finalizado o fallado en finalizar su operación de conexión de socket.
- boolean isReadable( )
  - Verifica si el canal se encuentra listo para leer.
- boolean isWritable( )
  - Verifica si el canal se encuentra listo para escribir.

Observaciones:

- Una key permanece válida hasta que sea cancelada haciendo uso del método cancel( ), cerrando el canal, o cerrando el selector.
- Al cancelar una key, esta no es removida inmediatamente del selector, en lugar de ello es agregado a un conjunto cancelled-key para eliminarlos durante la siguiente operación selection.

Registrando Canales

Para poder usar un canal con un selector, debe registrarse el canal en modo no bloqueado con el selector haciendo uso del siguiente método del canal seleccionable.

- SelectionKey register(Selector sel, int ops)

El segundo parámetro indicará qué eventos se escuchará haciendo uso del selector. Para ello se deben utilizar las siguientes constantes estáticas de la clase SelectionKey:

1. SelectionKey.OP_CONNECT
2. SelectionKey.OP_ACCEPT
3. SelectionKey.OP_READ
4. SelectionKey.OP_WRITE

Para conocer qué indica cada evento se puede usar de referencia los métodos recién presentados para SelectionKey.

Registrando Más de un Evento

Es posible registrar un canal con un selector e indicar que se esté alerta de más de una de los 4 eventos presentados.

Para ello es suficiente con realizar un bit-wise OR lógico a las opciones, obteniendo una nueva opción que será entregada como segundo parámetro en la llamada del método register( ).

Por ejemplo, si quisieramos estar pendientes de operaciones de lectura y escritura:

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

SelectionKey myKey = myChannel.register( mySelector, interestSet );
```

Seleccionando un Canal

Una vez se ha registrado uno o más canales podemos llamar a los métodos select( ), los cuales retornan la cantidad de keys que se encuentren “listas” en base a los eventos registrados.

- int select( )
  - Realiza una operación de bloqueo hasta que al menos una de las keys se encuentra lista para los eventos registrados.
  - Puede arrojar una IOException.
- int select(long timeout)
  - Realiza la operación bloqueada para un periodo de tiempo máximo indicado.
  - Puede arrojar una IOException.
- int selectNow( )
  - Realiza una operación no bloqueada y retorna de forma inmediata así no hayan keys listas.
  - Pude arrojar una IOException.

Recuperando las Keys

Una vez el método select( ) ha indicado que hay al menos una key lista, es posible acceder a ellas a través del método selectedKeys( ) de la clase Selector.

- Set<SelectionKey> selectedKeys( )
  - Retorna el conjunto selected-key del selector.
  - Las keys pueden ser retiradas pero no agregadas, cualquier intento de agregar una key generará un UnsopportedOperationException.

```
Set<SelectedKey> selectedKeys = mySelector.selectedKeys( );
```

Obs:

- Una key de selector y los conjuntos selected-keys no son en general seguros para uso en múltiples hilos concurrentes. Si un hilo fuera a modificar uno de estos conjuntos directamente, entonces el acceso debe ser controlado a través de una sincronización del conjunto mismo.

Iterando sobre las Keys

Para poder iterar sobre el conjunto selected-keys obtenido se puede utilizar un objeto iterator obtenido a partir del conjunto.

```
Set<SelectionKey> selectedKeys = selector.selectedKeys( );
Iterator<SelectionKey> keyIterator = selectedKeys.iterator( );

while( keyIterator.hasNext( ) ) {
   SelectionKey key = keyIterator.next( );
   if( key.isAcceptable( ) ) {
      // una conexión fu aceptada
   } else if (key.isConnectable()) {
      // una conexión fue establecida.
   } else if (key.isReadable()) {
      // un canal está listo para leer
   } else if (key.isWritable()) {
      // un canal está listo para escribir
   }
   keyIterator.remove( );
}
```

Obs:

- El iterator retornado es fail-fast: si el conjunto es modificado luego de que el iterator es creado, se arrojará un ConcurrentModificationException.

Recuperando Canales y Adjuntando objetos

Una vez recuperada la key, es posible obtener el canal registrado en el selector a partir de ella. Para ello se debe utilizar el siguiente método:

- ✓ SelectableChannel channel( )
  - Retorna el canal para el cual fue creado dicha key.
  - El método seguirá retornando el canal aún después de que la key sea cancelada.

Adicionalmente, es posible adjuntar objetos a una key y recuperarlos junto con la key en cada iteración. Para ello se tienen los siguiente métodos:

- ✓ Object attach(Object ob)
  - Adjunta el objeto dado a la key y retorna el objeto previamente adjuntado.
  - Sólo se puede adjuntar un objeto a la vez. Se puede adjuntar null para descartar el objeto anterior.
- ✓ Object attachment( )
  - Recupera el objeto adjuntado a la key actualmente.

Usando los Canales

Ya se vio cómo aceptar conexiones a través de un ServerSocketChannel. Resta ver cómo utilizar los SocketChannel que se hayan registrado.

Los SocketChannel pueden ser usado para las operaciones de lectura y escritura de la conexión. Para ello se cuenta con los siguientes métodos:

- ✓ int read(ByteBuffer dst)
  - ➢ Lee una secuencia de bytes desde el canal hacia el buffer dado.
  - ➢ Retorna el número de bytes leídos, o -1 si el canal a llegado al end-of-stream.
  - ➢ Puede arrojar un IOException.
- ✓ int write(ByteBuffer src)
  - ➢ Escribe una secuencia de bytes hacia el canal desde el buffer dado.
  - ➢ Retorna el número de bytes escritos.
  - ➢ Puede arrojar un IOException

El método wakeup( )

Un hilo que haya llamado al método select( ) que se encuentre bloqueado puede forzarse a dejar el método select incluso si no se han encontrado keys listas.

Para ello se debe tener otro hilo que llame al método wakeup( ) sobre el selector que haya causado el bloqueo.

- ✓ Selector wakeup( )
  - Ocasiona que la primera operación select que no haya retornado aún retorne inmediatamente.
  - Si no hay alguna operación select en progreso, la siguiente invocación de alguno de dichos métodos retornará inmediatamente, a excepción de SelectNow( ).
  - Invocar el método más de una vez entre operaciones de selección sucesivas tiene el mismo efecto que invocarlo sólo una vez.
  - Retorna el propio selector.