Aix-Marseille Université
Application Réseaux
2022

TD-5 : Programmation Sockets avec Java NIO

A. Channels et Buffers

1. Comment assurer la communication non-bloquante pour la lecture ou l'écriture dans un SocketChannel en Java NIO ? Quelles propriétés ont les objets de type SelectableChannel

```
SocketChannel sc =
   SocketChannel.open(new InetSocketAddress("127.0.0.1",3456));
sc.configureBlocking(false);
```

```
sc.isRegistered();     ==> false
sc.isConnected();     ==> true
sc.isBlocking();      ==> false
sc.isConnectionPending();  ==> false
sc.isOpen();               ==> true
sc.socket();  ==> Socket[addr=/127.0.0.1,port=3456,localport=54724]
sc.getRemoteAddress();  ==> /127.0.0.1:3456
sc.getLocalAddress();   ==> /127.0.0.1:54724
```

   (a) Mode : Blocking ou Non-blocking (méthode `isBlocking()` pour tester)

   (b) État : Enregistrer avec un selector ou pas (méthode `isRegistered()`)

2. Supposons qu'on ai un objet buffer de type ByteBuffer ayant pour valeurs internes : Position = 1, Limit = 3 et Capacity = 8. On exécute ensuite socketChannel.write(buffer) ; Combien de 'bytes' vont être transférés de buffer vers socketChannel ? Quelles seront les nouvelles valeurs de Position, Limit et Capacity ?
   — nb. de bytes transférés = 2
   — Position=3, Limit=3, Capacity=8

3. Quelles sont les différence entre les méthodes Buffer.clear() et Buffer.flip() pour un objet de type ByteBuffer. Expliquer avec un exemple précis.
   — Buffer.clear() : Position est mis à zero et Limit est mis égal à la capacité.
   — Buffer.flip() : Limit est mis en position actuel et Position est mis à zero.

```
> bb.clear();
==> java.nio.HeapByteBuffer[pos=0 lim=8 cap=8]
// contenu non effacé
> bb.flip();
==> java.nio.HeapByteBuffer[pos=0 lim=1 cap=8]
// limit = position; position = 0; contenu intact
```

B. Java NIO – Classe Selector

1. Décrire la classe Selector et la classe SelectionKey. Combien d'objets de type SelectionKey peuvent être associes à un Selector ?

   (a) A `Selector` is a multiplexor of SelectableChannel objects. A selector may be created by invoking the open method of this class. A selectable channel's registration with a selector is represented by a SelectionKey object. A key is added to a selector's key set as a side effect of registering a channel via the channel's register method.

   (b) A `SelectionKey` is created each time a channel is registered with a selector. A key remains valid until it is cancelled by invoking its cancel method, by closing its channel, or by closing its selector. Cancelling a key does not immediately remove it from its selector ; it is instead added to the selector's cancelled-key set for removal during the next selection operation. The validity of a key may be tested by invoking its isValid method.

   A selection key contains two operation sets represented as integer values : (1) The interest set determines which operation categories will be tested for readiness the next time one of the selector's selection methods is invoked. (2)The ready set identifies the operation categories for which the key's channel has been detected to be ready by the key's selector.

   (c) A selector maintains three sets of selection keys :
   - The key set contains the keys representing the current channel registrations of this selector.
   - The selected-key set is the set of keys such that each key's channel was detected to be ready for at least one of the operations identified in the key's interest set.
   - The cancelled-key set is the set of keys that have been cancelled but whose channels have not yet been deregistered.

2. Comment pourriez-vous enregistrer un SocketChannel à un Selector pour attendre les événements de types écriture (WRITE) ? Comment pourriez-vous enregistrer un ServerSocketChannel à un Selector pour attendre les demandes de connexion des clients ?

```
> sc.register(selector,SelectionKey.OP_WRITE);
> ssc.register(selector,SelectionKey.OP_ACCEPT);
```

3. On considère le code suivant pour la création d'un serveur et la connexion avec un premier client :

```
ServerSocketChannel␣ssc␣=␣ServerSocketChannel.open();
ssc.socket().bind(new␣InetSocketAddress(port));
SocketChannel␣csc␣=␣ssc.accept();
```

   Créer un seul Selector qui attend deux types d'événements : (1) les demandes de connexion des clients vers le serveur (ssc). (2) les événements de types écriture par le client (csc).

```
> ssc.configureBlocking(false);
> csc.configureBlocking(false);
> Selector selector = Selector.open();
> ssc.register(selector,SelectionKey.OP_ACCEPT);
> csc.register(selector,SelectionKey.OP_READ);
```

C. Concurrence en Java NIO

1. Est-il possible d'utiliser un seul objet Selector partagé par plusieurs threads ?

```
Oui. Les instances de Selector sont ‘‘thread-safe’’ : plusieurs threads
peuvent attendre sur le même objet. Attention : l’ensemble des clés ne
l’est par contre pas.
```

2. Quelle est l'utilité de la méthode Selector.wakeup() ? Quand avons-nous besoin de l'utiliser ?

```
Selector Selector.wakeup()
Causes the first selection operation that has not yet returned
to return immediately.
```

> Par exemple attend que la socket soit prête en écriture/lecture, mais un autre sait que cette socket ne sera plus utilisée

3. Les méthodes pour la classe Buffer, sont ils "thread-safe" (possible de les utiliser de manière concurrente) ? Si plusieurs threads partagent le même Buffer, que passe-t-il ?

> Non, ils ne sont pas “thread-safe”. Ils peuvent être partagés par plusieurs threads, ais il faut synchroniser les threads pour assurer la cohérence des écritures/lectures. (eg. éviter que plusieurs threads lisent la même donnée, ou écrivent dans la même case.).

4. Avec Java NIO, un seul Channel peut utiliser plusieurs objets Buffer pour la communication. Quelles sont les classes et méthodes à utiliser pour une communication (1) d'un Channel vers plusieurs Buffers, et (2) de multiples Buffers vers un seul Channel

> `SocketChannel` implémente `GatheringByteChannel` (écriture en provenance de plusieurs buffers) et `ScatteringByteChannel` (lecture dans plusieurs buffers)

   (a) `ScatteringByteChannel.read(ByteBuffer[] dsts)`

   (b) `GatheringByteChannel.write(ByteBuffer[] srcs)`