MEMO de TP

Programmation SOCKETS TCP/IP en JAVA

Ce document synthétise les principaux éléments des APIs JAVA pour la programmation de la communication via les mécanismes des sockets.

A- MODE CONNECTE :

1/ Classe java.net.Socket

Classe utilisée pour la programmation

- d'un client devant utiliser une connexion fiable TCP pour communiquer avec un processus serveur (distant).
- d'un serveur ayant acceptée une connexion TCP avec un client : socket utilisée pour la phase de transfert de données sur cette connexion.

a/ Principaux constructeurs :

```
public Socket(String host,int port) throws UnknownHostException,IOException
public Socket(InetAddress address, int port) throws IOException
public Socket(String host, int port, InetAddress localAddr, int localPort)throws
IOException
public Socket(InetAddress address, int port, InetAddress localAddr,
int localPort) throws IOException
```

Les deux premiers créent une socket directement connectée (via TCP) à la cible spécifiée sans spécifier les paramètres locaux alors que les deux derniers les imposent (binding).

Sémantique des paramètres:

`host` – le nom d'une machine distante, ou `null` pour l'adresse de loopback.
`port` – le numéro de port distant .
`address` – l'adresse IP de l'équipement distant.
`localAddr` – l'adresse IP locale à laquelle la socket doit être associée.
`localPort` - – le numéro de port local à laquelle la socket doit être associée.

Sémantique des exceptions :

`UnknownHostException` – si l'adresse IP de l'équipement n'a pu être déterminée.
`IOException` – si une erreur d'Entrée/Sortie se produit à la création de la socket.

b/ Principales méthodes :

```
public InputStream getInputStream() throws IOException
public OutputStream getOutputStream() throws IOException
```

Retournent respectivement un flux d'entrée et un flux de sortie associé à la socket courante. Habituellement, ces flux de données brutes (octets) sont liés à d'autres flux offrant davantage de fonctionnalités. Les opérations de lecture bloquante et d'écriture alors pouvant s'y appliquer.

Sémantique des exceptions :

`IOException` – si une erreur d'Entrée/Sortie se produit à la création du flux.

```
public void close()throws IOException
public InputStream shutdownInput() throws IOException
public OutputStream shutdownOutput() throws IOException
```

La première ferme la socket interdisant toute entrée et toute sortie et provoque la libération des ressources. Les secondes ne ferment respectivement que le flux d'entrée, ou de sortie, associée à la socket (aucune libération de ressource). Toute lecture, respectivement écriture, ultérieure lève une exception.

```
public InetAddress getInetAddress()
public InetAddress getLocalAddress()
public int getPort()
public int getLocalPort()
public SocketAddress getRemoteSocketAddress()
public SocketAddress getLocalSocketAddress()
public String toString()
```

Accesseurs permettant de récupérer des informations sur les extrémités de la connexion TCP liée à la socket ; la dernière méthode permettant la conversion de la socket en une chaîne de caractères et récapitulant ainsi les paramètres de la socket.

D'autres méthodes existent pour paramétrer finement le comportement de la connexion TCP (délais, buffers, types de flux –TOS, activité de connexion etc…)

Pour plus de détails :
http://docs.oracle.com/javase/6/docs/api/java/net/Socket.html

2/ Classe java.net. ServerSocket

Classe utilisée pour la programmation d'un serveur utilisant des connexions fiables TCP pour communiquer avec des processus clients (distants).

a/ Principaux constructeurs :

```
public ServerSocket(int port) throws IOException, BindException
public ServerSocket(int port, int size) throws IOException, BindException
public ServerSocket(int port, int size, InetAddress localAddr) throws
IOException, BindException
```

Sémantique des paramètres:

`port` – le numéro de port TCP local associé à la socket d'écoute .
`size` – nombre maximum de demandes stockables dans la FIFO (50 par défaut).
`localAddr` – l'adresse IP locale à laquelle la socket doit être associée.

Sémantique des exceptions :

`IOException` – si une erreur d'Entrée/Sortie se produit à la création de la socket. `BindException` – indisponibilité du port ou interdiction de son utilisation.

b/ Principales méthodes :

```
public Socket accept() throws IOException
```

Bloquante, elle retourne une nouvelle socket créée après acceptation d'une connexion demandée par un client distant. Cette nouvelle socket est dédiée au transfert des données sur cette connexion et s'utilise donc comme indiqué au paragraphe A-1-b.

Sémantique des exceptions :

`IOException` – si une erreur d'E/S se produit à la création de la nouvelle socket.

```
public void close()throws IOException
```

Fermeture de la socket d'écoute et libération du port utilisé.

```
public InetAddress getInetAddress()
public SocketAddress getLocalSocketAddress()
public int getLocalPort()
public String toString()
```

Accesseurs permettant de récupérer des informations sur la socket d'écoute du serveur; la dernière méthode permettant la conversion de la socket en une chaîne de caractères et récapitulant ainsi les paramètres de la socket.

D'autres méthodes existent pour ajuster des paramètres temporels régissant le comportement de la socket d'écoute.

Pour plus de détails :
http://docs.oracle.com/javase/6/docs/api/java/net/ServerSocket.html

B- MODE DATAGRAMME :

1/ Classe java.net.DatagramPacket

Classe utilisée pour manipuler une représentation d'un datagramme UDP.

a/ Principaux constructeurs :

```
public DatagramPacket(byte[] buf, int length, InetAddress address,int port)
public DatagramPacket(byte[] buf, int offset, int length,InetAddress address,
int port)
```

Construction d'un « objet paquet » pour l'envoi de `length` octets stockés dans `buf` – à partir du début ou depuis un déplacement d'`offset` octets – vers le port UDP spécifié de la machine distante dont l'adresse IP est donnée.

Sémantique des paramètres:

`buf` – les données à transmettre.
`length` – le nombre d'octets à expédier à partir du début, ou du déplacement
`offset` – le déplacement (depuis le début de buf) à partir d'où il faut expédier.
`port` – le numéro de port distant .
`address` – l'adresse IP de l'équipement distant.

```
public DatagramPacket(byte[] buf, int length)
public DatagramPacket(byte[] buf, int offset, int length)
```

Construction d'un « objet paquet » pour la réception de `length` octets à stocker dans `buf` – à partir du début ou depuis un déplacement d'`offset` octets.

Sémantique des paramètres:

`buf` – où stocker les données reçues.
`length` – le nombre d'octets à prélever
`offset` – le déplacement (depuis le début de buf) à partir d'où il faut stocker.

b/ Principales méthodes :

```
public InetAddress getAddress()
public int getPort()
public byte[] getData()
public int getlength()
```

Deux accesseurs permettant de récupérer l'adresse IP/numéro de port de la machine locale ou distante selon s'il s'agit d'un datagramme reçu ou créé par la machine locale.

Les deux suivants permettent respectivement de récupérer les données dont se compose le datagramme et le nombre d'octets correspondant.

Pour plus de détails :
http://docs.oracle.com/javase/6/docs/api/java/net/DatagramPacket.html

2/ Classe java.net.DatagramSocket

Classe utilisée pour réaliser réellement l'émission ou la réception d'un datagramme UDP.

a/ Principaux constructeurs :

```
public DatagramSocket() throws SocketException
public DatagramSocket(int port) throws SocketException
public DatagramSocket(int port, InetAddress localAddr) throws SocketException
```

Création – respectivement - d'une socket locale sur un port UDP anonyme (un client par exemple), ou sur un port local et écoutant sur toutes les interfaces réseau locales ou sur un port local n'écoutant que des datagrammes ayant pour destination l'adresse IP spécifiée (réduction à une seule interface réseau).

Sémantique des paramètres:

`port` – le numéro de port TCP local associé à la socket .
`localAddr` – l'adresse IP locale à laquelle la socket doit être associée.

Sémantique des exceptions :

`SocketException` – si une erreur ou impossibilité à l'ouverture du port.

b/ Principales méthodes :

```
public void send(DatagramPacket p) throws IOException
public void receive(DatagramPacket p) throws IOException
```

Permettent, respectivement ; l'envoi et la réception « bloquante » d'un datagramme.

Sémantique des exceptions :

`IOException` – si erreur lors de l'opération d'envoi ou de réception.

```
public void close()throws IOException
```

Fermeture de la socket et libération du port utilisé.
```
public int getLocalPort()
```

Accesseur permettant de récupérer le numéro de port local associé à la socket.

Pour plus de détails :
http://docs.oracle.com/javase/6/docs/api/java/net/DatagramSocket.html

C- ADRESSES et NOMS :

1/ Classe java.net.InetAdress

Classe utilisée pour manipuler des adresses IP et leurs associations aux noms d'hôtes. Pas de constructeurs publics.

```
public String getHostName()
```
retourne le nom de l'host ayant l'adresse IP courante.

```
public static InetAddress getByName(String host)throws UnknownHostException
```
détermine l'adresse IP d'un host à partir de son nom
```
public static InetAddress[] getAllByName(String host)throws UnknownHostException
```

détermine les adresses IP d'un host à partir de son nom
```
public static InetAddress getLocalHost()throws UnknownHostException
```

```
public static InetAddress getByAddress(String host, byte[] addr)throws
UnknownHostException
```
création d'un objet InetAdress constitué par un nom et une valeur d'adresse IP.

```
public byte[] getAddress()
```
retourne la valeur de l'adresse IP sous la forme d'un tableau d'octets.

```
public String getHostAddress()
```
retourne la valeur de l'adresse IP sous la forme d'une chaîne de caractères.

On peut également invoquer les méthodes :
```
public boolean equals(Object obj)
public String toString()
public int hashCode()
```

Pour plus de détails :
http://docs.oracle.com/javase/6/docs/api/java/net/InetAddress.html