Travaux Dirigés et Pratiques de Programmation Android n° 4

Client de chat

Les objectifs de ce TD sont :

- L'utilisation du thread principal et de « worker threads » ;
- L'établissement de connexions réseau et l'envoi de messages à travers celles-ci.

Pour cela, vous allez programmer un client de chat se connectant à un serveur tournant sur la machine projetée au vidéoprojecteur, en lui envoyant un login, puis en permettant d'envoyer des messages et de recevoir les messages des autres utilisateurs.

Mise en place

Télécharger le squelette de l'application que vous allez développer (https://www.lri.fr/~fiorenzi/Teaching/Android/TD4.zip). Tester l'application, et regarder le code fourni. L'application comporte deux activités :

- L'activité principale `TD4Activity` demande à l'utilisateur d'entrer un login, puis lance l'activité de chat `ChatActivity` lorsque ce login est non vide.
- L'activité `ChatActivity` permet à l'utilisateur d'écrire un message à envoyer aux autres clients du chat. Lorsque l'utilisateur appuie sur le bouton « ENTER », ce message est affiché dans la fenêtre de log, précédent du login de l'utilisateur. Cette activité n'utilise pour l'instant pas le réseau, et seul l'utilisateur de la tablette peut donc monologuer. Lorsque l'utilisateur appuie sur le bouton « LOGOUT », l'activité est fermée et on revient sur l'activité principale.

Le but du TP est de modifier cette activité afin de communiquer *via* le réseau, pour envoyer son login, ses messages, et recevoir ceux des autres personnes connectées.

Utilisation du réseau

Pour pouvoir utiliser et voir l'état du réseau, une application doit avoir les permissions correspondantes. Ajouter dans le manifeste, à l'intérieur des balises `manifest` (mais en dehors des balises `application`), les permissions pour l'utilisation du réseau et d'internet :

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Utilisation de flux sortant et entrant

Tout au long du TP, nous allons utiliser des flux de messages sortant et entrant. Nous allons d'abord étudier comment gérer ces flux.

Pour les flux sortant, nous allons utiliser la classe `PrintWriter` [1], offrant des méthodes de haut niveau pour envoyer des messages.

1. Ajouter un attribut à la classe `ChatActivity` de type `PrintWriter`, initialisé avec la sortie standard comme flux sortant :
   ```
   private PrintWriter writer = new PrintWriter(System.out, true);
   ```

[1] http://developer.android.com/reference/java/io/PrintWriter.html

2. À la fin de la méthode `onCreate`, afficher un message sur la sortie standard à l'aide de la méthode `void println (String str)`[2] de la classe `PrintWriter`.
3. Tester.

Pour les flux entrant, nous allons utiliser la classe `BufferedReader` [3], offrant notamment une méthode `String readLine ()`[4] permettant de lire en une seule fois une ligne complète envoyée sur le flux sortant.

1. Ajouter un attribut à la classe `ChatActivity` de type `BufferedReader`, initialisé avec l'entrée standard comme flux entrant :

   ```
   private BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
   ```

Lancement d'un nouveau thread et connexion au serveur

On ne peut pas accéder aux ressources lentes telles que le réseau dans le thread principal, dont le but est de gérer l'interface graphique. On va donc lancer un nouveau thread au démarrage de l'activité `ChatActivity` qui nous servira à ouvrir la connexion réseau.

Dans un premier temps, on va se contenter de mettre en place ce thread, afin d'afficher un message sur l'écran de la tablette.

1. Au sein de la classe `ChatActivity`, définir une classe privée `StartNetwork` héritant de la classe `AsyncTask<Void, Void, Boolean>` [5]:
   - le premier type `Void` indique que ce thread ne prend pas de paramètres lors de son lancement ;
   - le deuxième type `Void` indique que ce thread ne renverra rien au cours de son exécution ;
   - le troisième type, `Boolean`, indique que ce thread renverra un Booléen à la fin de son exécution.
2. Dans cette classe `StartNetwork`, surcharger la méthode `protected Boolean doInBackground(Void... v)` afin de toujours renvoyer la valeur `false` (on modifiera le code de cette méthode dans un deuxième temps). Noter que, comme indiqué dans le type de la classe, cette méthode prend une liste de `Void` en argument et renvoie un `Boolean`.
3. Toujours dans cette classe, surcharger la méthode `protected void onPostExecute(Boolean b)` de manière à afficher dans la zone de chat (à l'aide de la méthode `displayMessage` de la classe `ChatActivity`) le message « Connected to server » si `b` est vrai, et « Could not connect to server » sinon.
4. Surcharger la méthode `protected void onStart()`, en n'oubliant pas d'appeler la méthode de la super-classe. Compléter cette méthode de manière à créer un nouvel objet de la classe `StartNetwork`, puis à lancer son exécution (sans argument puisqu'on a choisi que le thread ne prenne pas de paramètres) : `new StartNetwork().execute();`
5. Tester.

On va maintenant ouvrir la connexion réseau dans ce thread, en modifiant le code de la méthode `doInBackground`.

1. Ouvrir une socket, à l'aide du constructeur `Socket (String dstName, int dstPort)`[6] de la classe `Socket`[7]. Les arguments de ce constructeurs identifient le serveur :

[2] http://developer.android.com/reference/java/io/PrintWriter.html#println(java.lang.String)
[3] http://developer.android.com/reference/java/io/BufferedReader.html
[4] http://developer.android.com/reference/java/io/BufferedReader.html#readLine()
[5] http://developer.android.com/reference/android/os/AsyncTask.html

   - `dstName` est son nom ou adresse IP : pour le TP, `massy.chadok.info` ;
   - le port sur lequel il écoute : pour le TP, 7878.
2. Renvoyer `true` si l'ouverture de la socket s'est correctement effectuée, et `false` sinon.
3. Tester.
4. Que se passe-t-il si l'on met l'ouverture de socket directement dans la méthode `onStart` ?
5. Que se passe-t-il si l'on met l'appel à `displayMessage` dans la méthode `doInBackground` ?

Communication avec le serveur

Vous avez maintenant une connexion ouverte pour pouvoir communiquer avec le serveur. Le protocole de communication de ce dernier est le suivant :

- Lorsqu'il reçoit le message « LOGIN login », où « login » est une chaîne de caractères quelconque, il connecte l'utilisateur correspondant avec son login, et envoie à tous les utilisateurs connectés le message « Welcome [login] ».
- Lorsqu'il reçoit le message « LOGOUT », il déconnecte l'utilisateur correspondant, et envoie à tous les utilisateurs connectés le message « Bye bye [login] ».
- Lorsqu'il reçoit le message « SEND message », où « message » est un message quelconque, il envoie à tous les utilisateurs connectés « [login]: [message] ».

Envoi du login

Dans un premier temps, vous allez envoyer votre login au serveur de chat. Cela peut se faire dès l'ouverture de la socket : compléter la méthode `doInBackground` de la manière suivante.

1. Récupérer le flux sortant de la socket dans la variable définie au-dessus :
   ```
   writer = new PrintWriter(socket.getOutputStream(), true);
   ```
2. Envoyer dans ce flux un message de login :
   ```
   writer.println("LOGIN " + login);
   ```
3. Tester et observer au vidéoprojecteur.

Envoi de messages et déconnexion

De manière similaire, utiliser le flux sortant pour :

- Envoyer au serveur un message de type « LOGOUT » lorsque l'utilisateur appuie sur le bouton « LOGOUT » ;
- Envoyer au serveur les messages de l'utilisateur lorsqu'il appuie sur « SEND ».

Tester et observer au vidéoprojecteur.

Récupération des messages envoyés par le serveur

Il reste à récupérer les messages envoyés pour le serveur, notamment afin d'afficher sur la tablette les messages des autres utilisateurs du client de chat. Pour cela, il faut en permanence écouter sur le flux entrant de la socket pour savoir lorsqu'un message est reçu. Nous allons donc utiliser un deuxième thread géré également par une classe héritant de `AsyncTask` [8], mais pour lequel c'est la progression qui va nous intéresser et non le résultat final.

[6] http://developer.android.com/reference/java/net/Socket.html#Socket(java.lang.String, int)
[7] http://developer.android.com/reference/java/net/Socket.html
[8] http://developer.android.com/reference/android/os/AsyncTask.html

1. Là où l'on récupère le flux sortant de la socket, récupérer également le flux entrant dans l'attribut `reader` :
   ```
   reader = new BufferedReader(new
   InputStreamReader(socket.getInputStream ()));
   ```
2. Définir une nouvelle classe privée `ReadMessages` héritant de `AsyncTask`.
3. Dans cette classe, surcharger la méthode `protected Void doInBackground(Void... v)` afin de faire une boucle infinie qui va :
   - lire une ligne du `reader` :
     `String message = reader.readLine();`
   - publier ce message durant sa progression :
     `publishProgress(message);`

   Si une exception est lancée, sortir de la boucle. Finir cette méthode par `return null;` afin qu'elle ait bien `Void` comme type de retour.
4. Toujours dans la classe `ReadMessages`, surcharger la méthode `protected void onProgressUpdate(String... messages)` afin d'afficher le message reçu en cours de progression (là encore, grâce à la méthode `displayMessage`).
   Rappel : la syntaxe `String... messages` indique de `messages` est un tableau d'éléments de type `String`. Dans notre cas, on n'y a mis qu'un seul élément, en position 0.
5. Quel est le meilleur moment pour exécuter ce thread ? À l'endroit choisi, créer un nouvel objet de la classe `ReadMessages` et lancer son exécution.
6. Tester en se loguant et en envoyant des messages.
7. Que se passe-t-il lorsqu'on essaie de se déconnecter ? Pourquoi ? Remédier au problème grâce aux méthodes `boolean cancel (boolean mayInterruptIfRunning)`[9] et `boolean isCancelled()`[10] appelées aux bons endroits de l'activité. Tester.

Utilisation de la classe Thread

Remplacez les classes héritant de `AsyncTask` par des classes héritant de `Thread`. Tester.

Pour faire le TP chez vous

Vous pouvez exécuter vous-même le serveur en ligne de commande (https://www.lri.fr/~fiorenzi/Teaching/Android/TD4Serveur.jar). On peut le lancer avec la ligne de commande suivante : `java -jar TD4Serveur.jar` (sous Linux ou MacOS, dans un terminal ; sous Windows, dans l'invite de commande). Le port par défaut sur lequel il écoute est le 7777. Le serveur doit tourner sur la même machine que l'émulateur ; son adresse IP est alors `10.0.2.2`.

Si vous avez un message d'erreur au lancement du serveur, c'est sans doute que le port 7777 est déjà occupé. Utilisez alors un autre port au lancement, comme par exemple : `java -jar TD4Serveur.jar 7878`. Il faut alors penser à utiliser le port correspondant dans votre application cliente.

À vous de jouer

Écrire une application implantant un serveur de chat pour ce protocole, gérant dans un premier temps un unique client, avant de gérer plusieurs clients. Voici quelques étapes pour vous guider. La fin de cette partie vous indique comment tester votre serveur (ce que vous devez faire tout au long des diverses questions).

[9] http://developer.android.com/reference/android/os/AsyncTask.html#cancel(boolean)
[10] http://developer.android.com/reference/android/os/AsyncTask.html#isCancelled()

1. Définir une interface qui pourra afficher la connexion et déconnexion des utilisateurs, ainsi que les messages reçus. On pourra s'inspirer de la boîte `ScrollView` et de la méthode `displayMessage` de l'activité `ChatActivity`.
2. Créer un thread qui ouvre une socket serveur sur le port 7777. On pourra utiliser la méthode `ServerSocket (int port)`[11] de la classe `ServerSocket` [12]. Ne pas oublier d'autoriser les accès au réseau dans le manifeste.
3. Dans ce thread, attendre une connexion sur le port 7777, et afficher un message dans l'interface lorsqu'il y en a une. On pourra utiliser la méthode `Socket accept ()`[13] de la classe `ServerSocket`.
4. Gérer les échanges avec le client.

   Dans un premier temps, se contenter d'afficher sur la tablette tous les messages envoyés par le client. Dans un deuxième temps, réagir aux actions du client : attendre qu'il envoie un message de type « LOGIN », puis attendre (indéfiniment) des messages de type « SEND » et « LOGOUT ». Faire l'affichage correspondant (en publiant ces messages au fur et à mesure de la progression du thread).

   Pour interpréter les messages envoyés par le client, on pourra s'aider des méthodes auxiliaires suivantes :

```
private String getLogin(String message) {
    Scanner scan = new Scanner(message);
    scan.useDelimiter(" ");
    if (scan.hasNext() && scan.next().equals("LOGIN") && scan.hasNext()) {
        return scan.next();
    } else {
        return null;
    }
}

private String getSendOrLogout(String message) {
    Scanner scan = new Scanner(message);
    scan.useDelimiter(" ");
    if (scan.hasNext() && scan.next().equals("SEND")) {
        return message.substring(5);
    } else {
        return null;
    }
}
```

   La méthode `getLogin` renvoie le login s'il s'agit bien d'une commande de type « LOGIN login », et `null` sinon. La méthode `getSendOrLogout` renvoie le message de l'utilisateur s'il s'agit d'une commande « SEND message », et `null` s'il s'agit de toute autre commande (et donc notamment d'une commande de type « LOGOUT »).

5. Faire en sorte qu'à chaque fois que le client se déconnecte, le serveur se mette à attendre un nouveau client pour relancer un protocole de chat.
6. Rendre le serveur robuste aux clients ne respectant pas le protocole : dans ce cas, il doit rejeter le client (éventuellement en lui fournissant une erreur), puis attendre un nouveau client. On pourra utiliser la méthode `void close ()`[14] de la classe `Socket`.
7. Ajouter un bouton permettant d'arrêter et de redémarrer le serveur.

[11] http://developer.android.com/reference/java/net/ServerSocket.html#ServerSocket(int)
[12] http://developer.android.com/reference/java/net/ServerSocket.html
[13] http://developer.android.com/reference/java/net/ServerSocket.html#accept()
[14] http://developer.android.com/reference/java/net/Socket.html#close()

8. Gérer plusieurs clients, en conservant une liste de tous les clients connectés. Il faudra nécessairement utiliser la classe `Thread` pour pouvoir gérer les clients en parallèle.

Pour tester

À chaque fois que vous lancez l'émulateur ou que vous connectez votre tablette/téléphone, vous devez rediriger son port 7777 vers le port 7777 de votre machine (vous pouvez adapter le numéro du port si vous en utilisez un autre).

- Sur les machines de l'IUT :
  1. Ouvrir l'invite de commande.
  2. Entrer `c:`
  3. Entrer `cd Android/sdk/platform-tools`
  4. Entrer `adb.exe forward tcp:7777 tcp:7777`
- Sous Windows :
  1. Ouvrir l'invite de commande.
  2. Entrer `cd AppData/Local/Android/sdk/platform-tools` (ou un autre chemin si vous n'utilisez pas celui par défaut).
  3. Entrer `adb.exe forward tcp:7777 tcp:7777`
- Sous Linux et MacOS :
  1. Ouvrir un terminal.
  2. Entrer `cd Android/Sdk/platform-tools` (ou un autre chemin si vous n'utilisez pas celui par défaut).
  3. Entrer `./adb forward tcp:7777 tcp:7777`

Vous pouvez utiliser votre client tournant dans un autre émulateur. Cependant, avoir deux émulateurs ouverts risque de fortement ralentir votre ordinateur. Pour éviter cela, on vous propose un client en ligne de commande (https://www.lri.fr/~fiorenzi/Teaching/Android/TD4Client.jar), à lancer de la même manière que le serveur en ligne de commande ci-dessus. Pour utiliser ce client, entrez d'abord votre login, puis vos messages, et enfin le message « LOGOUT » pour quitter.

Pour tester la robustesse de votre serveur, on vous fournis également un client plus simple (https://www.lri.fr/~fiorenzi/Teaching/Android/TD4ClientSimple.jar) envoyant au serveur ce que vous entrez tel quel. Cela vous permet aussi bien de respecter le protocole (en envoyant « LOGIN login », puis « SEND message », jusqu'à « LOGOUT ») que de ne pas le respecter.

Pour aller plus loin

Revenir sur le client.

1. Que se passe-t-il lorsqu'on tourne la tablette ? Gérer ce cas pour avoir le comportement attendu. On pourra se référer à ce tutoriel : http://developer.android.com/training/basics/activity-lifecycle/recreating.html
2. Faire en sorte que le message tapé par l'utilisateur soit envoyé lorsque ce dernier appuie sur la touche « entrée » du clavier, en complément du bouton « ENTER ».