Le bluetooth sous android

Nicolas Fleurot

20 mai 2013

Table des matières

Chapitre 1

Description

1.1 Le bluetooth

Le bluetooth et un type de connexion par onde radio dont le fonctionnement d'un point de vue programation se rapproche du TCP d'un point de vue ordre et corruption de données. Ces spécificité technique ne seront pas aborder, voir wikipedia pour plus de détaille.

1.2 ce cours

Ce cours a pour but d'apprendre les bases de l'utilisation du bluetooth sur android, les different bout de code présenté seront souvent à adapter. de plus, certaine fonction sont fictive et ne sont la que pour representer une suite d'instruction possible. Enfin, certain "état" telle que *REQUEST_ENABLE_BT* sont a définir soit meme en tant qu'attribut constant de la class.

Il est fortement recommander de garder un oeil sur le projet d'exemple en meme temps que vous liser ce cours, pour suivre ce qu'il se passe dans un vrai projet et en comprendre le fonctionnement.

1.3 divers

1.3.1 Structure propre au java

Le java utilise parfois une structure assez special consistant a déclarer un "listenner" puis a mettre entre accolade la fonction qu'il appel juste derriere, avant le point virgule, on peu voir comme exemple le *BroadcastReceiver*

1.3.2 Socket

Un socket représente une interface réseau. un socket TCP ou bluetooth pourrai s'imaginer comme un goblet attacher a un fil, chaque correspondant aillant son propre goblet, le but étant de retrouver par quelle fil les goblet sont connecter.

1.3.3 Toast

Toast permet d'afficher de petit popup de notification, il s'utilise sous cette forme :

```
1 Toast.makeText(Context context, CharSequence text, int duration).show();
```

Avec context l'activiter dans lequelle il apparait (en general, this), text le texte à afficher et duration la durée d'apparition (generalement Toast.LENGTH_SHORT)

1.3.4 intent

Les intent sont la glue qui séparre les differente activiter, il permetent d'appeler des activiter et de recuperer leur code de sortie. Il permetent aussi de passer des info entre les activiter.

1.3.5 Thread

Une thread en quelque sorte un processus vivant en parrallele d'un programme, et qui permet donc de faire des action simultané. ce qui se revele tres utile pour certain calcule ou pour effectuer des action bloquante, c'est a dire qui bloque l'execution de la thread dans laquelle vie le programme pour X raison.

Chapitre 2

Mise en place

2.1 récupérer l'adaptateur

Pour pouvoir utilié une connexion bluetooth, il faut déja commencér par récupérer le prériphérique bluetooth de l'appareil. Il représente le socket de connexion, c'est par lui que passeron toutes les données.

```
1 // on recupere le l'adaptateur.
2 mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
3
4 // si l'adaptateur est null, le peripherique ne supporte pas le bluetooth
5 if (mBluetoothAdapter == null) {
6     // on affiche un message sur l'ecran de l'utilisateur.
7     Toast.makeText(this, "Bluetooth indisponnible", Toast.LENGTH_LONG).show();
8 return;
9 }
```

2.2 Activer l'adaptateur

Une fois l'adaptateur récupéré, il faut vérifier qu'il soit utilisable, et donc activé.

```
1 // si le bluetooth n'est pas activer
2 if (!mBluetoothAdapter.isEnabled()) {
3 // on affiche une fenetre pre-cree permettant sont activation
4     Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
5     startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
6 }
```

2.3 Liste des péripherique

Il faudrai au préalable crée un widget permettant l'affichage de la liste des périphérique, mais ce serai trop s'éloigner du sujet, ce ne sera donc pas vu dans ce cours.

2.3.1 Récupérer la liste des périphérique connu

Récupérer la liste des périphérique connu se fait de manière tres simple puisqu'il existe déjà une méthode l faisant à notre plaçe :

```
1 // on d\'eclare une collection qui stockera la liste
2 Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();
3 // si on a detecte quelque chose
4 if(pairedDevices > 0) {
5     for(BluetoothDevice device : pairedDevices) {
6         // on ajoute les peripherique a la liste / au widget / etc...
7         mArrayAdapter.add(device.getName() + "\n" + device.getAdress());
8     }
9 }
```

2.3.2 Découvrire des périphériques

La découverte de périphérique est assé complexe. il faut commencer par appeler la méthode *startDiscovery()* qui renvois immédiatement un booleen pour savoir si le périphérique a commencer la détéction.

Une fois la détéction lancé, il faut déclarer un objet de type BroadcastReceiver qui se chargera de récuperer les différent périphériques disponnible

```
// on declare le recever
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
    public void onReceive(Contect context, Intent intent) {
        // on stock l'"intention" du peripherique
        String action = intent.getAction();
        // si le peripherique "voulais" etre decouvert
        if(BluetoothDevice.ACTION_FOUND.equals(action)){
            // on recupere le peripherique
            BluetoothDevice device =
            intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            // on l'ajoute a la liste
            mArrayAdapter.add(device.getName() + "\n" + device.getAdress());
        }
    }
};
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(mReceiver, filter);
```

2.4 Rendre un périphérique découvrable

Cela se fait de la même maniere que d'activer le bluetooth :

```
// on utilise un outil precree
Intent discoverableIntent =
new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
// on peu optionnelement specifier le temps de discoverabiliter
discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
// on lance l'activiter
startActivity(discoverableIntent);
```

Chapitre 3

Utilisation

3.1 Connexion

Le bluetooth utilise une architecture client / serveur, il faut donc un socket client et un socket serveur. Comme chaque péripherique doit écouté les connexions, une méthode d'implementation et de préparer chaque péripherique comme un serveur a l'écoute de connexion, ainsi, des qu'une connexion est recu, l'un des péripherique devient client et l'autre serveur automatiquement. Une autre implementation serai de définir explicitement chaque périphérique comme étant client ou serveur.

Les méthode des socket étant bloquante, il faut les executer dans des thread séparé.

3.1.1 coté serveur

on commence par définir une nouvelle class implementant Thread :

```
1 private class AcceptThread extends Thread {
2     ...
3 }
```

La class n'a qu'un seul attribut, le socket serveur :

```
1 private final BluetoothServerSocket mmServerSocket;
```

Le constructeur permet de recuperer un socket serveur unique correspondant a l'application :

```
1  public AcceptThread(){
2      // on utilise un socket temporaire car mmServerSocket et null
3      BluetoothServerSocket tmp = null;
4      // on tente de recuperer le socket serveur unique
5      try {
6          tmp = mBluetoothAdapter.listenUsingRfcommWithServiceRecord(NAME, UUID);
7      } catch (IOException e) {
8          ...
9      }
10     mmServerSocket = tmp;
11 }
```

La méthode permettant de récupérer une connexion et assez simple, elle se contente d'écouter jusqu'à ce qu'une connexion ou une erreur arrive :

```
1  public void run() {
2      BluetoothSocket socket = null;
3      while(true){
4          // on gere les \'eventuelle erreurs
5          try {
6              socket = mmServerSocket.accept();
7          } catch(IOException e){
8              break;
9          }
10         // si on a reussi a obtenir une connexion
11         if(socket != null){
12             manageConnectedSocket(socket);
13             // si l'on en veut qu'une seul connexion
14             mmServerSocket.close();
15             break;
16         }
17     }
18 }
```

manageConnectedSocket() est une méthode fictive senser representer l'initialisation d'une nouvelle thread se servant du socket fraichement recuperer pour envoyer et recevoir des données

Finalement, une fois la ou les connexions établie, il faut arreter la thread :

```
public void cancel(){
    try{
        mmServerSocket.close();
    } catch(IOException e){
        ...
    }
}
```

3.1.2 coté client

on commence par définir une nouvelle class implementant Thread :

```
private class ConnectThread extends Thread {
    ...
}
```

La class a deux attribut, le socket client et un bluetoothDevice représentant celui que nous avons recupéré précédement :

```
private final BluetoothSocket mmSocket;
private final BluetoothDevice mmDevice;
```

Le constructeur permet de recuperer un socket unique correspondant a l'application :

```
public ConnectThread(BluetoothDevice device){
    // on utilise un socket temporaire car mmServerSocket et null
    BluetoothServerSocket tmp = null;
    mmDevice = device
    // on tente de recuperer le socket unique
    try {
        tmp = device.createRfcommSocketToServiceRecord(UUID);
    } catch (IOException e) {
        ...
    }
    mmSocket = tmp;
}
```

La méthode permettant de récupérer une connexion et assez simple, elle se contente d'écouter jusqu'à ce qu'une connexion ou une erreur arrive :

```
public void run() {
    // on annule une \'eventuelle recherche pouvant ralentire la connexion
    mBluetoothAdapter.cancelDiscovery();
    try {
        // methode bloquante permettant de connecter le p\'eripherique par le
            socket
        mmSocket.connect();
    } catch (IOException connectException) {
        // si on arrive pas a se connecter, on ferme le socket et on quitte la
            fonction
        try {
            mmSocket.close();
        } catch (IOException closeException) {
            ...
        }
        return;
    }
    // si on a reussi a se connecter
    manageConnectedSocket(mmSocket);
}
```

manageConnectedSocket() est une méthode fictive senser representer l'initialisation d'une nouvelle thread se servant du socket fraichement recuperer pour envoyer et recevoir des données

Finalement, une fois la ou les connexions établie, il faut arreter la thread :

```
public void cancel(){
    try{
        mmSocket.close();
    } catch(IOException e){
        ...
    }
}
```

3.2 Lecture / Ecriture

Pour utilisé un périphérique bluetooth, il faut re-implanté la classe Thread :

```
private class ConnectedThread extends Thread {
    ...
}
```

Il faut aussi lui mettre des attribut, corespondant au socket bluetooth et au flux entrant/sortant :

```
private final BluetoothSocket mmSocket;
private final InputStream mmInStream;
private final OutputStream mmOutStream;
```

Toute class a un constructeur, celle la prend en paramètre un socket bluetooth et initialise les flux avec objet temporaire (encapsulation, toussa toussa) :

```
public ConnectedThread(BluetoothSocket socket) {
    mmSocket = socket;

    // on cree les flux temporaire
    InputStream tmpIn = null;
    OutputStream tmpOut = null;

    // on tente de recuperer les flux d'entree sortie du socket bluetooth
    try {
        tmpIn = Socket.getInputStream();
        tmpOut = Socket.getOutputStream();
    } catch(IOException e)
    {
        ...
    }

    // on initialise les flux de la class avec les flux temporaire
    mmInStream = tmpIn;
    mmOutStream = tmpOut;
}
```

3.2.1 Écriture

Écrire et donc envoyer des donné par l'intermédiaire du bluetooth est extremement simple, il suffit d'implementer une méthode qui prend en paramètre un tableau d'octet et qui se chargera de l'envoyer au flux de sortie :

```
public void write(byte[] bytes){
    // on tente d'ecrire les donnees
    try{
        mmOutStream.write(bytes);
    } catch(IOException e){
        ...
    }
}
```

3.2.2 Lecture

La lecture de données est beaucoup plus complexe, la thread tourne en continue à l'ecoute de données (c'est d'ailleur l'utilité de la thread, l'écoute et l'écriture étant bloquante, elles doivent s'éxecuter séparement), il va donc falloire une variable de la taille d'un byte, permettant de recuperer une a une toute les données, un buffer pour cette même récuperation, et executer tout sa en boucle jusqu'à ce que le programme s'arrête ou qu'il y ai une erreur :

```
1  public void run(){
2      // notre buffer
3      byte[] buffer = new byte[1024];
4      int bytes;
5
6      while(true) {
7          try {
8              // on lit le flux
9              bytes = mmInStream.read(buffer);
10             /* on envoie les donnees la ou elle doivent aller */
11         } catch(IOException e) {
12             break;
13         }
14     }
15 }
```

3.3 Néttoyage

Il faut toujour tout nettoyer avant de quitter une application, le garbage collector fait une grosse partie du travaille, mais il ne peu s'attaquer à de la mémoire encore réfférencé, il faut donc le faire avant de quitter :

```
1  public void cancel() {
2      try{
3          mmSocket.close();
4      } catch(IOException e) {
5          // on peu afficher un message d'erreur ou ecrire dans d'eventuel log
6      }
7  }
```