

Le bluetooth sous android

Nicolas Fleurot

19 mai 2013

Table des matières

1	Description	2
1.1	Le bluetooth	2
1.2	ce cours	2
1.3	divers	2
2	Mise en place	3
2.1	recupérer l'adaptateur	3
2.2	Activer l'adaptateur	3
2.3	Liste des peripherique	3
2.3.1	Recupérer la liste des périphérique connu	3
2.3.2	Découvrir des périphériques	4
2.4	Rendre un périphérique découvrable	4
3	Utilisation	5
3.1	Connexion	5
3.1.1	coté serveur	5
3.1.2	coté client	6
3.2	Lecture / Ecriture	7
3.2.1	Écriture	7
3.2.2	Lecture	7
3.3	Nettoyage	8

Chapitre 1

Description

1.1 Le bluetooth

1.2 ce cours

1.3 divers

Chapitre 2

Mise en place

2.1 récupérer l'adaptateur

Pour pouvoir utiliser une connexion bluetooth, il faut déjà commencer par récupérer le périphérique bluetooth de l'appareil. Il représente le socket de connexion, c'est par lui que passeront 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 indisponible", 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 son activation
4     Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
5     startActivityResult(enableIntent, REQUEST_ENABLE_BT);
6 }
```

2.3 Liste des périphérique

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

2.3.1 Récupérer la liste des périphériques connus

Récupérer la liste des périphériques connus se fait de manière très simple puisqu'il existe déjà une méthode qui fait à notre place :

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

2.3.2 Découvrir des périphériques

La découverte de périphérique est assés complexe. il faut commencer par appeler la méthode *startDiscovery()* qui renvoie immédiatement un boolean 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 récupérer les différent périphériques disponible

```
1 // on declare le receiver
2 private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
3     public void onReceive(Context context, Intent intent) {
4         // on stock l'"intention" du peripherique
5         String action = intent.getAction();
6         // si le peripherique "voulais" etre decouvert
7         if (BluetoothDevice.ACTION_FOUND.equals(action)) {
8             // on recupere le peripherique
9             BluetoothDevice device =
10                 intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
11             // on l'ajoute a la liste
12             mArrayAdapter.add(device.getName() + "\n" + device.getAdress());
13         }
14     }
15 };
16 IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
17 registerReceiver(mReceiver, filter);
```

2.4 Rendre un périphérique découvrable

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

```
1 // on utilise un outil precree
2 Intent discoverableIntent =
3 new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
4 // on peu optionnelement specifier le temps de discoverabiliter
5 discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
6 // on lance l'activer
7 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 périphérique doit écouter les connexions, une méthode d'implémentation et de préparer chaque périphérique comme un serveur à l'écoute de connexion, ainsi, dès qu'une connexion est reçue, l'un des périphériques devient client et l'autre serveur automatiquement. Une autre implémentation serait de définir explicitement chaque périphérique comme étant client ou serveur.

Les méthodes des sockets étant bloquantes, il faut les exécuter dans des threads séparés.

3.1.1 côté serveur

on commence par définir une nouvelle classe implémentant Thread :

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

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

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

Le constructeur permet de récupérer un socket serveur unique correspondant à l'application :

```
1 public AcceptThread() {  
2     // on utilise un socket temporaire car mmServerSocket est null  
3     BluetoothServerSocket tmp = null;  
4     // on tente de récupérer 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 est 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 gère les \eventuelles erreurs  
5         try {  
6             socket = mmServerSocket.accept();  
7         } catch (IOException e){  
8             break;  
9         }  
10        // si on a réussi à obtenir une connexion  
11        if(socket != null){  
12            manageConnectedSocket(socket);  
13            // si l'on en veut qu'une seule connexion  
14            mmServerSocket.close();  
15            break;  
16        }  
17    }  
18 }
```

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

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

```
1 public void cancel(){
2     try{
3         mmServerSocket.close();
4     } catch(IOException e){
5         ...
6     }
7 }
```

3.1.2 coté client

on commence par définir une nouvelle class implementant Thread :

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

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

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

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

```
1 public ConnectThread(BluetoothDevice device){
2     // on utilise un socket temporaire car mmServerSocket et null
3     BluetoothServerSocket tmp = null;
4     mmDevice = device
5     // on tente de recuperer le socket unique
6     try {
7         tmp = device.createRfcommSocketToServiceRecord(UUID);
8     } catch (IOException e) {
9         ...
10    }
11    mmSocket = tmp;
12 }
```

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     // on annule une \eventuelle recherche pouvant ralentire la connexion
3     mBluetoothAdapter.cancelDiscovery();
4     try {
5         // methode bloquante permettant de connecter le p\eripherique par le
6         // socket
7         mmSocket.connect();
8     } catch (IOException connectException) {
9         // si on arrive pas a se connecter, on ferme le socket et on quitte la
10        // fonction
11        try {
12            mmSocket.close();
13        } catch (IOException closeException) {
14            ...
15        }
16        return;
17    }
18    // si on a reussi a se connecter
19    manageConnectedSocket(mmSocket);
20 }
```

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

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

```
1 public void cancel(){
2     try{
3         mmSocket.close();
4     } catch(IOException e){
5         ...
6     }
7 }
```

```

6     }
7 }

```

3.2 Lecture / Ecriture

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

```

1 private class ConnectedThread extends Thread {
2     ...
3 }

```

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

```

1 private final BluetoothSocket mmSocket;
2 private final InputStream mmInStream;
3 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) :

```

1 public ConnectedThread(BluetoothSocket socket) {
2     mmSocket = socket;
3
4     // on cree les flux temporaire
5     InputStream tmpIn = null;
6     OutputStream tmpOut = null;
7
8     // on tente de recuperer les flux d'entree sortie du socket bluetooth
9     try {
10         tmpIn = Socket.getInputStream();
11         tmpOut = Socket.getOutputStream();
12     } catch (IOException e) {
13     }
14     ...
15 }
16
17 // on initialise les flux de la class avec les flux temporaire
18 mmInStream = tmpIn;
19 mmOutStream = tmpOut;
20 }

```

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 :

```

1 public void write(byte[] bytes){
2     // on tente d'ecrire les donnees
3     try{
4         mmOutStream.write(bytes);
5     } catch (IOException e){
6     }
7     ...
8 }

```

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'exécuter séparément), il va donc falloir une variable de la taille d'un byte, permettant de recuperer une a une toute les données, un buffer pour cette même récupération, 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 Nettoyage

Il faut toujours tout nettoyer avant de quitter une application, le garbage collector fait une grosse partie du travail, mais il ne peut s'attaquer à de la mémoire encore référencée, il faut donc le faire avant de quitter :

```

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

```