Tutorial 1

USO DE BLUETOOTH EN JAVA

Vicente Cantón Paterna

14/04/2015

En este tutorial se pretende mostrar cómo implementar el servicio de Bluetooth en aplicaciones para Smartphone con Android.

INTRODUCCIÓN

Este primer tutorial trata sobre el uso de la tecnología Bluetooth en aplicaciones Android, para lo cual se va a ver una aplicación de ejemplo. No se va a explicar ahora el fundamento teórico del Bluetooth, pero está resumido aquí. El código completo de la aplicación puede descargarse al final del tutorial.

Para no extender demasiado el tutorial se parte de la base de que el Bluetooth está activado y vinculado con el dispositivo con el que se quiere conectar, pero se puede programar la aplicación para que busque dispositivos. Además no se van a comentar todos los métodos y clases de la aplicación de ejemplo, sino las relacionadas con la utilización de Bluetooth. En cualquier caso están comentadas en el propio código.

Resumen de uso general de Bluetooth

Los pasos para utilizar Bluetooth en Android son los siguientes:

- Se instancia e inicializa la clase BluetoothAdapter para obtener el adaptador de Bluetooth con getDefaultAdapter().
- Se obtienen los dispositivos vinculados con getBondedDevices().
- Se establece el dispositivo de los dos va a actuar como servidor y el que va a actuar como cliente.
- El servidor ha de extender la clase *Thread*, crear un BluetoothServerSocket mediante listenUsingRfcommWithServiceRecord() y escuchar peticiones en el método *run* a través de bluetoothserversocket.accept().
- El cliente ha de extender la clase *Thread*, crear un BluetoothSocket utilizando el BluetoothDevice que actúa como servidor y el UUID de la aplicación mediante el método createRfcommSocketToServiceRecord() y en el método *run* llamar a la clase connect().
- Cuando el cliente se conecte al servidor, ambos dispondrán de un socket BluetoothSocket con el que manejar la conexión. Ya solo queda instanciar las clases InputStream y OutputStream para la lectura y escritura, respectivamente.

Ejemplo-tutorial de una aplicación que usa Bluetooth

En la mayoría de aplicaciones lo normal es que la aplicación con Bluetooth quiera hacer de servidor o de cliente, según el caso, por lo que primero se va a crear la clase BluetoothService que va a contener al *thread* del cliente, el servidor y la conexión en sí misma. De esta forma es más fácil manejarlo desde la actividad principal.

La clase BluetoothService va a estar ejecutándose constantemente en segundo plano, por lo que hay que hacer que extienda a la clase *Thread* para implementar el método *run*. Además, al estar ejecutándose en segundo plano, se necesita alertar a la actividad principal de los mensajes recibidos a través de Bluetooth. Una forma de hacerlo es a través de la clase *Handler*. Un *Handler* se crea en la actividad principal y se le pasa a la actividad en segundo plano (en este caso BluetoothService) para que avise a la clase principal cuando ocurra un evento. Más información aquí.

BluetoothService

La clase BluetoothService tiene el siguiente constructor:

```
public BluetoothService(Handler handler, BluetoothAdapter bAdapter){

    this.handler = handler;
    this.bAdapter = bAdapter;

}
```

La clase BluetoothAdapter es necesaria para realizar la conexión, ya que es la que permite obtener el adaptador Bluetooth del dispositivo para activar/desactivar el Bluetooth, iniciar conexiones…

Dentro de la misma clase BluetoothService, para facilitar su uso, se incluyen las clases BluetoothServer, BluetoothClient y BluetoothConnection. La primera tiene la función de poner a la aplicación en modo de escucha de peticiones para que BluetoothClient pueda establecer la conexión. Una vez establecida, se maneja el envío y recepción de información desde BluetoothConnection.

Además son necesarios 2 métodos dentro de BluetoothService que actúan como intermediarios entre los objetos anteriores y el hilo principal:

```
public synchronized void iniciarConexion(){

    // Si se esta intentando realizar una conexion mediante un hilo cliente,
    // se cancela la conexion

    if(bClient != null)
    {
        bClient.cancelarConexion();
        bClient = null;
    }

    // Si existe una conexion previa, se cancela
    if(bConnect != null)
    {
        bConnect.cancelarConexion();
        bConnect = null;
    }

    // Arrancamos el hilo servidor para que empiece a recibir peticiones
    // de conexion
    if(bServer == null)
    {
        bServer = new BluetoothServer();
        bServer.start();
    }
}

// Instancia un hilo conector
public synchronized void solicitarConexion(BluetoothDevice dispositivo)
{

    if(bClient != null)
    {
        bClient.cancelarConexion();
        bClient = null;
    }


    // Si existia una conexion abierta, se cierra y se inicia una nueva
    if(bConnect != null)
    {
        bConnect.cancelarConexion();
        bConnect = null;
    }

    // Se instancia un nuevo hilo conector, encargado de solicitar una conexion
    // al servidor, que sera la otra parte.
    bClient = new BluetoothClient(dispositivo);
    bClient.start();

}
```

BluetoothServer

Puesto que se va a estar escuchando peticiones, se ha de extender nuevamente *Thread*. Se ha de crear la clase BluetoothServerSocket, que a través del BluetoothAdapter anterior genera un socket para escuchar peticiones:

```
private BluetoothServerSocket serverSocket;

public BluetoothServer(){

    BluetoothServerSocket tmpServerSocket = null;

    // Creamos un socket para escuchar las peticiones de conexion
    try {
        tmpServerSocket = bAdapter.listenUsingRfcommWithServiceRecord("AppEjemplo",
                UUID.fromString("00001101-0000-1000-8000-00805f9b34fb"));
    } catch(IOException e) {
        e.printStackTrace();
    }

    serverSocket = tmpServerSocket;
}
```

El UUID es un identificador universal. En el caso de Bluetooth se utiliza para distinguir entre aplicaciones que puedan estar utilizando Bluetooth.

Una vez que ya se tiene creado el socket del servidor, se implementa el método *run* encargado de escuchar y aceptar peticiones entrantes a través del método *accept():*

```
public void run() {
    BluetoothSocket socket = null;
    while (true) {
        try {
            socket = serverSocket.accept();
        } catch (IOException e) {
            break;
        }
        if (socket != null) {
            realizarConexion(socket);
            try {
                serverSocket.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
            break;
        }
    }
}
```

Cuando se acepte una petición se llama al método realizarConexion, cuyo parámetro es BluetoothSocket. Este método es el encargado de crear la clase BluetoothConnection con el dispositivo remoto que realizó la petición de conexión:

```
public synchronized void realizarConexion(BluetoothSocket socket)
{
    bConnect = new BluetoothConnection(socket);
    bConnect.start();
}
```

Este método ha de ser llamado tanto desde el dispositivo que actúa como servidor como desde el dispositivo que hace de cliente para iniciar la conexión, tal como se verá a continuación.

BluetoothClient

La clase BluetoothClient también ha de extender a *Thread* y tiene como objetivo realizar la petición de conexión al servidor. Para ello es necesario que el cliente sepa el dispositivo y el UUID de la aplicación a la que quiere conectarse:

```
public class BluetoothClient extends Thread {

    private static final String TAG = "En cliente: ";
    private final BluetoothSocket mmSocket;
    private final BluetoothDevice device;
    private UUID uuid;

    public BluetoothClient(BluetoothDevice device){

        BluetoothSocket tmp = null;
        this.device = device;
        uuid = UUID.fromString("00001101-0000-1000-8000-00805f9b34fb");

        // Se genera el socket para el BluetoothDevice seleccionado

        try {
            tmp = device.createRfcommSocketToServiceRecord(uuid);
        } catch (IOException e) {
            e.printStackTrace();
        }

        mmSocket = tmp;
    }
```

En este caso se llama al método createRfcommSocketToServiceRecord(UUID) ya que lo que se quiere hacer es crear una petición en vez de estar escuchando.

Ahora falta implementar el método *run*:

```
public void run(){

    if (bAdapter.isDiscovering())
        bAdapter.cancelDiscovery();

    try {
        mmSocket.connect();
    } catch (IOException e) {
        e.printStackTrace();
        Log.e(TAG,"Error abriendo el socket");
        try {
            mmSocket.close();
        } catch (IOException inner) {
            Log.e(TAG, "Error cerrando el socket", inner);
        }
    }

    // Reiniciamos el hilo cliente, ya que no lo necesitaremos mas
    synchronized(BluetoothService.this)
    {
        bClient = null;
    }

    // Realizamos la conexion
    realizarConexion(mmSocket);
}
```

El método *run* de BluetoothClient es similar al de BluetoothServer. Dado el BluetoothSocket creado en el constructor se intenta realizar la conexión con el servidor. Si es capaz de establecerse, se borra el hilo del cliente y se llama al método realizarConexion(BluetoothSocket) para crear el hilo que maneja la conexión.

BluetoothConnection

Se encarga de mandar y recibir mensajes. Necesita, por tanto, las clases InputStream y OutputStream para la lectura y escritura, respectivamente. Además recibe como parámetro el BluetoothSocket de la conexión ya establecida:

```
private class BluetoothConnection extends Thread {

    private final BluetoothSocket socket;           // Socket
    private final InputStream inputStream;  // Flujo de entrada (lecturas)
    private final OutputStream outputStream;    // Flujo de salida (escrituras)

    public BluetoothConnection(BluetoothSocket socket){

        this.socket = socket;

        setName(socket.getRemoteDevice().getName() + "[ "+socket.getRemoteDevice().getAddress());

        InputStream tmpInputStream = null;
        OutputStream tmpOutputStream = null;

        try {
            tmpInputStream = socket.getInputStream();
            tmpOutputStream = socket.getOutputStream();
        } catch (IOException e) {
            e.printStackTrace();
        }
        inputStream = tmpInputStream;
        outputStream = tmpOutputStream;
        handler.sendEmptyMessage(MSG_CONEXION);
    }
```

Se deben obtener los *stream* de entrada y salida a partir del objeto BluetoothSocket del constructor. Además, en este caso, se envía desde el *handler* a la actividad principal un mensaje vacío MSG_CONEXION. En el hilo principal está codificado para cambiar del *layout* encargado de buscar los dispositivos bluetooth vinculados con el móvil al *layout* encargado de enviar mensajes y mostrar los mensajes recibidos.

Por último tenemos el método *run* encargado de aceptar los mensajes que llegan mediante el método *read*():

```
// Metodo principal del hilo, encargado de realizar las lecturas
public void run()
{
    byte[] buffer = new byte[1024];
    int bytes;
    // Mientras se mantenga la conexion el hilo se mantiene en espera ocupada
    // leyendo del flujo de entrada
    while(true)
    {
        try {
            // Leemos del flujo de entrada del socket
            bytes = inputStream.read(buffer);

            // Enviamos la informacion a la actividad a traves del handler.
            // El metodo handleMessage sera el encargado de recibir el mensaje
            // y mostrar los datos recibidos en el TextView
            handler.obtainMessage(MSG_RECIBIR, bytes, -1, buffer).sendToTarget();
        }
        catch(IOException e) {
            e.printStackTrace();
        }
    }
}
```

El método obtainMessage de *handler* indica el tipo de mensaje (MSG_RECIBIR), el número de bytes leídos, y dos argumentos de tipo *object*. El primero, que no se va a utilizar, se establece a -1. El segundo contiene un vector de bytes con los bytes recibidos.

Por último sólo queda crear un método para mandar mensajes a través del outputStream y para cancelar la conexión:

```
public void escribir(byte[] buffer)
{
    try {
        // Escribimos en el flujo de salida del socket
        outputStream.write(buffer);

    }
    catch(IOException e) {
        e.printStackTrace();
    }
}

public void cancelarConexion()
{
    try {
        // Forzamos el cierre del socket
        socket.close();
    }
    catch(IOException e) {
        e.printStackTrace();
    }
}
```

Cómo utilizar la clase de ejemplo BluetoothService en aplicaciones que necesiten Bluetooth

Cuando se requiera el uso de Bluetooth, basta con instanciar una clase de tipo BluetoohService y de inicializarla con un *Handler*, para realizar la comunicación entre el hilo principal y la clase BluetoothService; y con BluetoothAdapter para poder establecer un dispositivo como servidor y otro como cliente. Después hay que llamar a bluetoothService.iniciarConexion() para actuar como servidor y a bluetoothService.conectarDispositivo(dispositivo.getAdress()) para actuar como cliente. A partir de aquí ya se puede llamar al método enviar(byte[] mensaje) dentro de BluetoothService para enviar un mensaje. A la hora de leer, el Handler es el encargado de obtener el mensaje y llevarlo al hilo principal.

A modo de ejemplo se ha creado una aplicación que hace uso de esta clase, que puede encontrarse en el siguiente repositorio de Github: ejemploBluetooth

Por último dejo también un enlace a una serie de tutoriales para el uso de Bluetooth en el que me he basado y he utilizado para aprender cómo funciona: tutoriales bluetooth