

MINI TUTO PARA CONEXIONES

CLIENTE - SERVIDOR CON J2ME

LEANDRO FLÓREZ ARISTIZÁBAL

xxxlepexxx@gmail.com

CONEXIONES CLIENTE SERVIDOR

Server

Se necesita un objeto de tipo `StreamConnection` para crear la conexión.

```
StreamConnection conexion;
```

Adicionalmente se pueden usar objetos de tipo `InputStream` o `DataInputStream` y `OutputStream` o `DataOutputStream` para la recepción y envío de datos respectivamente.

```
InputStream is; //DataInputStream dis;  
OutputStream os; //DataOutputStream dos;
```

Una vez se desea hacer la conexión, esta debe hacerse en otro hilo para no bloquear la aplicación. Las conexiones del tipo RFCOMM ó SPP (Serial Port Profile) son bidireccionales y orientadas a streams y están basadas en las interfaces `StreamConnectionNotifier` y `StreamConnection`. Un servidor se crea llamando a `Connector.open()` con el String apropiado para la conexión servidor. Este String empieza con el prefijo de protocolo "btspp://". Un objeto de tipo `StreamConnectionNotifier` es retornado cuando se abre una conexión servidora y es su método `acceptAndOpen()` el que acepta conexiones de clientes remotos.

```
StreamConnectionNotifier service =  
(StreamConnectionNotifier)Connector.open("btspp://localhost:uuidServicio; name=nombreServicio");  
  
conexion= (StreamConnection) service.acceptAndOpen(); //Espera una conexión de un cliente.
```

*localhost: indica que va a ser un servidor.

*Si la conexión desde el cliente va a ser directa (o sea que no se hará búsqueda de servicios sino que el cliente intentará conectarse al servidor especificando su dirección y un canal de comunicación) necesitaremos saber que canal asignó el celular servidor al servicio para que desde el cliente se especifique este mismo canal.

```
ServiceRecord record = LocalDevice.getLocalDevice().getRecord(service);  
//Se guarda el url del servicio para imprimirlo y saber cual es el puerto que se le asignó al servicio  
String url = record.getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT, true);
```

Cuando se crea la conexión, se pueden enviar y recibir datos haciendo uso de los objetos `OutputStream` o `DataOutputStream` e `InputStream` o `DataInputStream` respectivamente y hacer uso de sus métodos `write()` & `read()`.

```
os = conexion.openOutputStream(); // dos=conexion.openDataOutputStream();  
is=conexion.openInputStream(); // dis=conexion.openDataInputStream();
```

Client

Se necesita un objeto de tipo `StreamConnection` para crear la conexión.

```
StreamConnection conexion;
```

Adicionalmente se pueden usar objetos de tipo `InputStream` o `DataInputStream` y `OutputStream` o `DataOutputStream` para la recepción y envío de datos respectivamente.

```
InputStream is; //DataInputStream dis;  
OutputStream os; //DataOutputStream dos;
```

Una vez se desea hacer la conexión, esta debe hacerse en otro hilo para no bloquear la aplicación. Las conexiones del tipo RFCOMM ó SPP (Serial Port Profile) son bidireccionales y orientadas a streams y están basadas en las interfaces `StreamConnectionNotifier` y `StreamConnection`. Un cliente se crea llamando a `Connector.open()` con el String apropiado para la conexión cliente. Este String empieza con el prefijo de protocolo "btspp://". Un objeto de tipo `StreamConnection` es retornado cuando se abre una conexión cliente (Cuando se hace la petición al servidor).

```
conexion = (StreamConnection) Connector.open( "btspp://direcciónServidor:canalServicio");
```

* 0000000decaf es un ejemplo de dirección del servidor.

*1 es un ejemplo de canal que el servidor puede asignar al servicio.

Cuando se crea la conexión, se pueden enviar y recibir datos haciendo uso de los objetos `OutputStream` o `DataOutputStream` e `InputStream` o `DataInputStream` respectivamente y hacer uso de sus métodos `write()` & `read()`.

```
os = conexion.openOutputStream(); // dos=conexion.openDataOutputStream();  
is=conexion.openInputStream(); // dis=conexion.openDataInputStream();
```

BÚSQUEDA DE DISPOSITIVOS

Client

Se necesita un objeto de tipo LocalDevice y otro de tipo DiscoveryAgent. El primero representa al dispositivo local (Bluetooth del dispositivo). Para obtener la única instancia existente de esta clase llamaremos al método getLocalDevice() de la clase LocalDevice. El segundo objeto es único y lo obtendremos a través del método getDiscoveryAgent() del objeto LocalDevice:

```
public LocalDevice dispositivoLocal;  
public DiscoveryAgent da;  
...  
dispositivoLocal = LocalDevice.getLocalDevice();  
da = dispositivoLocal.getDiscoveryAgent();
```

Para comenzar una nueva búsqueda de dispositivos llamaremos al método startInquiry() del objeto DiscoveryAgent. Este método requiere dos argumentos. El primer argumento es un entero que especifica el modo de conectividad que deben tener los dispositivos a buscar. Este valor deberá ser DiscoveryAgent.GIAC o bien DiscoveryAgent.LIAC. El segundo argumento es un objeto que implemente DiscoveryListener. A través de este último objeto serán notificados los dispositivos que se vayan descubriendo. Para cancelar la búsqueda usaremos el método cancelInquiry().

```
da.startInquiry(int modoConectividad, DiscoveryListener claseQueEscucha);
```

* DiscoveryAgent.GIAC es un ejemplo de modo de conectividad.

* Si la clase que se encarga de escuchar los eventos que ocurren durante la búsqueda es la misma en la que se ejecuta (la búsqueda), el argumento será "this" (sin las comillas ""), de lo contrario será un objeto de la clase que implementa DiscoveryListener.

La clase que implemente DiscoveryListener deberá implementar los métodos propios de la interfaz. Estos son:

```
public void deviceDiscovered(RemoteDevice dispositivoRemoto, DeviceClass claseDispositivo)  
public void inquiryCompleted(int motivoFinalización)
```

* Cada vez que se descubre un dispositivo se llama al método deviceDiscovered(). Nos pasa dos argumentos. El primero es un objeto de la clase RemoteDevice que representa el dispositivo encontrado. El segundo argumento nos permitirá determinar el tipo de dispositivo encontrado.

* inquiryCompleted() es llamado cuando la búsqueda de dispositivos ha finalizado. Nos pasa un argumento entero indicando el motivo de la finalización. Este argumento podrá tomar los valores: DiscoveryListener.INQUIRY_COMPLETED si la búsqueda ha concluido con normalidad, DiscoveryListener.INQUIRY_ERROR si se ha producido un error en el proceso de búsqueda, o DiscoveryListener.INQUIRY_TERMINATED si la búsqueda fue cancelada.

REGISTRO DE SERVICIOS

Server

Lo primero de todo, para ofrecer un servicio a través de Bluetooth necesitaremos poner nuestro dispositivo en modo visible. Esto se hace, recordemos, a través de la clase LocalDevice:

```
LocalDevice dispositivoLocal = LocalDevice.getLocalDevice();  
dispositivoLocal.setDiscoverable(DiscoveryAgent.GIAC);
```

Para crear una conexión servidora necesitaremos pasarle una URL al método Connector.open(). La URL deberá comenzar por "btspp://localhost:". Además del host de la URL deberemos indicar el UUID que identifica el servicio. Posteriormente indicaremos el nombre del servicio y otros parámetros, como por ejemplo el requerimiento o no de autenticación.

Al pasar una URL de este tipo al método Connector.open(), éste nos devolverá un "notifier", que en el caso de SPP será un objeto StreamConnectionNotifier. Este objeto nos permitirá escuchar conexiones entrantes de los clientes.

```
UUID uuid = new UUID(0xABCD);  
String nombre = "Ejemplo SPP";  
StreamConnectionNotifier notifier;
```

```
notifier = (StreamConnectionNotifier) Connector.open(("btspp://localhost:"+uuid.toString()+";name="+nombre);
```

Una vez hemos creado el "notifier" que escuchará las conexiones clientes debemos especificar los atributos de nuestro servicio. Los atributos de nuestro servicio están almacenados en un ServiceRecord. El ServiceRecord se obtiene a través de la clase LocalDevice con el método getRecord(). Una vez tenemos el ServiceRecord podremos establecer sus atributos mediante el método setAttributeValue() al que le pasaremos un identificador numérico y un objeto DataElement que representará el valor del atributo de servicio.

```
ServiceRecord rec = dispositivoLocal.getRecord(notifier);
```

```
//Modifico la descripción del servicio  
rec.setAttributeValue(0x0101,new DataElement(DataElement.STRING,"Que gran juego! Es para dos personas!"));  
  
//Modifico atributo 0x100 o sea el nombre del servicio Ejemplo SPP por Juego SPP  
rec.setAttributeValue(0x0100,new DataElement(DataElement.STRING,"Juego SPP"));
```

Llegados a este punto ya podemos escuchar conexiones clientes a través de acceptAndOpen().

```
StreamConnection conn = notifier.acceptAndOpen();
```

BÚSQUEDA DE SERVICIOS

Client

Se necesita un objeto de tipo LocalDevice y otro de tipo DiscoveryAgent. El primero representa al dispositivo local (Bluetooth del dispositivo). Para obtener la única instancia existente de esta clase llamaremos al método `getLocalDevice()` de la clase LocalDevice. El segundo objeto es único y lo obtendremos a través del método `getDiscoveryAgent()` del objeto LocalDevice:

```
public LocalDevice dispositivoLocal;  
public DiscoveryAgent da;  
  
...  
dispositivoLocal = LocalDevice.getLocalDevice();  
da = dispositivoLocal.getDiscoveryAgent();
```

Para comenzar una nueva búsqueda de servicios llamaremos al método `searchServices()` del objeto DiscoveryAgent. Este método requiere dos argumentos. El primero es un array de enteros con el que especificaremos los atributos de servicio en los que estamos interesados. En la interfaz ServiceRecord, los servicios son descritos a través de atributos que son identificados numéricamente. Este array contendrá los ID de estos atributos. El segundo argumento es un array de identificadores de servicio. Nos permite especificar los servicios en los que estamos interesados. El tercer argumento es el dispositivo remoto sobre el que vamos a realizar la búsqueda. Por último pasaremos un objeto que implemente DiscoveryListener que será usado para notificar los eventos de búsqueda de servicios.

```
da.searchServices(int[] attrSet, UUID[] uuidSet, RemoteDevice dispositivoRemoto, DiscoveryListener claseQueEscucha)
```

Cada servicio tiene varios atributos como nombre, descripción, etc., y si estamos interesados en conocer uno o más de ellos, deberemos especificarlos en la búsqueda agregándolos en el arreglo attrSet, de lo contrario podemos usar null teniendo en cuenta que si luego queremos saber por ejemplo el nombre del servicio indicando su atributo, nos arrojará un NullPointerException ya que en la búsqueda no lo especificamos.

Si deseamos saber todos los servicios del puerto Serie especificamos su UUID 0x1101, pero si deseamos un servicio propio en especial, escribiremos su UUID como por ejemplo "86b4d249fb8844d6a756ec265dd1f6a3", false. Ejemplo:

```
da.searchServices(new int[]{0x0100,0x0101}, new UUID[]{new UUID(0x1101)}, dispositivo_remoto, this);
```

La clase que implemente DiscoveryListener deberá implementar los métodos propios de la interfaz. Estos son:

```
servicesDiscovered(int transID, ServiceRecord[] servRecord)  
serviceSearchCompleted(int transID, int respCode)
```

Cada que se descubre un servicio se llama a `servicesDiscovered()` con dos argumentos; el primero (transID) identifica el proceso de búsqueda para saber cual fue la que lo invocó ya que se pueden hacer búsquedas simultáneas y el segundo (servRecord) es un array de objetos ServiceRecord. Un objeto ServiceRecord describe las características de un servicio bluetooth, en otras palabras es el servicio como tal.

`serviceSearchCompleted()` es llamado cuando termina la búsqueda de servicios. Consigo vienen dos argumentos que indican cual fue el proceso de búsqueda que finalizó (transID) y el motivo (respCode) por el cual terminó.