Hálózati kommunikáció

HTTP kapcsolatok kezelése

Balogh Tamás

balogh.tamas@autsoft.hu

Tartalom

- Rövid és hosszabb távú kapcsolatok bemutatása (NFC, Bluetooth, UDP, TCP/IP socket)
- HTTP kapcsolatok kezelése
- Tipikus adatformátumok és azok feldolgozási lehetőségei
- Aszinkron kommunikáció helyes kezelése
- REST API-k hatékony használata
- Hatékony hálózati kommunikációs könyvtárak

Rövid és hosszabb távú kapcsolatok

NFC, Bluetooth, UDP, TCP/IP socket

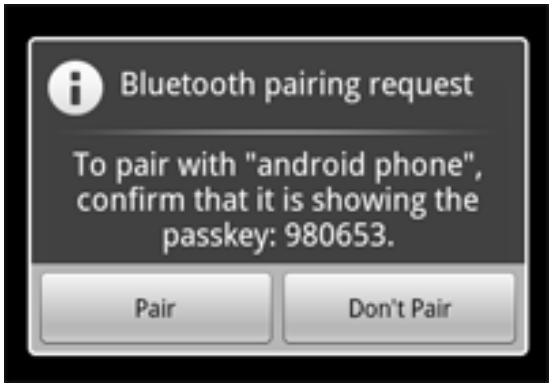
Bluetooth

- Bluetooth készülékek felderítése
- Helyi Bluetooth adapter-ek lekérdezése a párosított eszközökhöz
- RFCOMM csatorna kiépítése
- Adattovábbítás készülékek közt
- Több egyidejű kapcsolat kezelése
- Fontosabb osztályok:
  - android.bluetooth csomag

Kapcsolódás

- Socket megnyitása:

```
val btSocket = device.
createRfcommSocketToServiceRecord(MY_UUID)
```



- MY_UUID:
  - Egyedi azonosító, amit a saját szolgáltatásunk azonosítására használhatunk
- A párosítást a rendszer kezeli, addig vár a kapcsolat megnyitással, amíg nem történik meg a párosítás.
- **Ne felejtsük el megfelelően bezárni a kapcsolatokat!**

Kapcsolat fogadása

```
val serverSocket = bluetoothAdapter.
    listenUsingRfcommWithServiceRecord(SERVICE_NAME, MY_UUID)
val socket = serverSocket.accept()
```

- Az *accept()* hívás addig blokkolódik, míg nem jön kapcsolódási kérés
- Sikeres kapcsolódást követően a kommunikáció az ismert *InputStream*-mel és *OutputStream*-mel történik
- Ne felejtsük el a végén bezárni a socketeket!

Bluetooth low energy

- Android 4.3 (API Level 18)
- Bluetooth Low Energy; *central role* és *peripheral* APIk
- Felderítés és kommunikáció támogatása (karakterisztikák írása)
- Lényegesen kevesebb energiahasználat (telefon és eszköz oldalon)
- Tipikus eszközök:
  - Közelség érzékelők, szívritmus szenzorok, eü. kiegészítők, stb.

Near Field Communication (NFC)

- Rövidtávú vezeték nélküli kommunikációs technológia
- <4cm távolságon belül működik
- NFC tag és mobil telefon közti kis méretű adatátvitel (payload)
- Mobil telefon és mobil telefon közti kis méretű adatátvitel
- NFC Forum által meghatározott formátum: NDEF (NFC Data Exchange Format)

NFC Tag-ek jellemzői

- Írható/olvasható/egyszer írható
- Komplex Tag-ek tartalmazhatnak matematikai műveleteket is és lehet külön kriptográfia hardver egységük authentikáció céljából
- Még bonyolultabb Tag-ek akár saját működési környezettel is rendelkezhetnek és egyszerűbb kód végrehajtására is alkalmasak

TCP/IP Socket

- Szabványos *Socket* implementáció
- Jól ismert *java.net.Socket* osztály a kapcsolatok megnyitására
- *java.net.ServerSocket* osztály a bejövő kapcsolatok fogadására
  - Localhoston az alkalmazások egymás közti kommunikációja is megoldható
- InputStream és OutputStream támogatás az adatok olvasására és írására

Socket példa

```
val socket = Socket("192.168.2.112",8787)
val is = socket.getInputStream()
val isr = InputStreamReader(is, "UTF-8")
val resultBuffer = StringBuilder()
var inChar: Int = isr.read()
while (inChar != -1) {
  resultBuffer.append(inChar as Char)
  inChar = isr.read())
}
val result = resultBuffer.toString()
// result kezelése
// ...
is.close()
socket.close()
```

UDP Kommunikáció

- User Datagram Protocol (UDP) a Datagram üzenetek támogatására
- Kicsi, gyors üzenetek kezelése
- UDP nem garantálja hogy megérkezik a csomag
- Tipikusan akkor amikor a csomagvesztés nem kritikus
- Például:
  - Multimédia stream
  - Játékok
  - Stb...

UDP Androidon

- *java.net.DatagramSocket*: socket
- Broadcast üzenet esetén:

```
val datagramSocker = DatagramSocket()
datagramsocker.broadcast = true
```

- *java.net.DatagramPacket*: UDP osztály

UDP üzenetek küldése

```
val message = "UDP Test"

val socket = DatagramSocket()

// Broadcast esetén

socket.broadcast = true

val localAddress =
        netAddress.getByName("192.168.0.110")

val messageBytes = msg.bytes

val packet = DatagramPacket(message, msg.length,
localAddress, 10100);

socket.send(p)
```

UDP üzenet fogadása

```
val message = ByteArray(1500)
val packet = DatagramPacket(message,message.length)
val socket = DatagramSocket(10100)
socket.receive(packet)
val msgStr = String(message, 0, packet.length)
socket.close()
```

HTTP Kapcsolatok kezelése

HTTP kommunikáció Android platformon

- Egyik leggyakrabban használt kommunikációs technológia
- HTTP metódusok
  - GET, POST, PUT, DELETE
- Teljes körű HTTPS támogatás és certificate import lehetőség
- REST kommunikáció támogatása (Representational State Transfer)

HTTP kapcsolatok kezelése

- Szükséges engedély:
  <uses-permission android:name=
  "android.permission.INTERNET"/>
- Új szálban kell megvalósítani a hálózati kommunikáció hívást!
- Ellenőrizzük a HTTP válasz kódot:
  - http://www.restpatterns.org/HTTP_Status_Codes
  - http://www.w3.org/Protocols/HTTP/HTRESP.html
- HTTP REST
  - http://en.wikipedia.org/wiki/Representational_state_transfer
- Ügyeljünk az alapos hibakezelésre
- HTTP GET példa:
  - http://avalon.aut.bme.hu/~tyrael/phpget.php?name=JohnDoe

HTTP könyvtárak Android-on

- A rendszer két megvalósítás is tartalmaz:
  - Standard Java HTTP implementáció (***HttpURLConnection***)
  - **Apache** HTTP implementáció (*HttpClient*)
- Apache Depricated – Ne használjuk, ki is vették
- Igazán egyik sem jó
  - 3rd party megoldás – Square OkHttp
  - http://square.github.io/okhttp/

HTTP GET - HttpURLConnection

```
fun httpGet(urlAddr: String) {
    var reader: BufferedReader? = null
    try {
        val url = URL("http://mysrver.com/api/getitems")
        val conn = url.openConnection() as HttpURLConnection
        reader = BufferedReader(InputStreamReader(conn.inputStream))
        var line: String?
        do {
            line = reader.readLine()
            System.out.println(line)
        } while (line != null)
    } catch (e: IOException) {
        e.printStackTrace()
    } finally {
        if (reader != null) {
            try {
                reader.close()
            } catch (e: IOException) {
                e.printStackTrace()
            }
        }
    }
}
```

HTTP POST- HttpURLConnection

```
fun httpPost(urlAddr: String, content: ByteArray) {
    // ...
    var os: OutputStream? = null
    try {
        val url = URL("http://mysrver.com/api/refreshitems")
        val conn = url.openConnection() as HttpURLConnection
        conn.requestMethod = "POST"
        conn.doOutput = true
        conn.useCaches = false
        os = conn.outputStream
        os.write(content)
        os.flush()
        // …
    } catch (e: IOException) {
        e.printStackTrace()
    } finally {
        // …
        if (os != null) {
            try {
                os.close()
            } catch (e: IOException) {
                e.printStackTrace()
            }
        }
    }
}
```

HTTP GET - OkHttp

```
val okhttpClient = OkHttpClient.Builder()
        .connectTimeout(5000, TimeUnit.MILLISECONDS)
        .build()

val request = Request.Builder()
        .url("http://api.open-notify.org/astros.json")
        .build()


val response = okhttpClient.newCall(request).execute()
val code = response.code()
val body = response.body()!!.string()
```

Példa

- OkHttp – Get példa

Tipikus adatformátumok

Adatok küldése, válaszok feldolgozása

- Sokszor egy előre definiált formátumban/protokollon történik a kommunikáció kliens és szerver között
- Legtöbb esetben egy harmadik fél szerverétől kapott válasz is valamilyen jól strukturált formátumban érkezik
- Tipikus formátumok:
  - CSV (Comma Separated Value(s))
  - JSON (JavaScript Object Notation)
  - XML (Extensible Markup Language)
- Természetesen lehet saját protokoll is

JSON formátum

- Szintaktikai elemek: ’{’, ’]’ ,’[’, ’]’ ,’:’ ,’,’
- Példa:

```
{
   "keresztnev" : "Elek",
   "vezeteknev" : "Teszt",
   "kor" : 23,
   "cim" :
   {
      "utca" : "Baross tér",
      "varos" : "Budapest",
      "iranyitoszam" : "1087"
   },
   "telefon":
   [
      {
        "tipus"  : "otthoni",
        "szam": "123 322 1234"
      },
      {
        "tipus"  : "mobil",
        "szam": "626 515 1567"
      }
   ]
}
```

JSON feldolgozás

- *JSONObject*:
  - JSON objektumok parse-olása
  - Elemek elérhetősége a kulcs megadásával:
    - `getString(String name)`
    - `getJSONObject(String name)`
    - `getJSONArray(String name)`
  - JSON objektum létrehozása *String*-ből vagy *Map*-ból
- *JSONArray*:
  - *JSONObject*-hez hasonló működés JSON tömbökkel
  - Parse-olás, elemek lekérdezése index alapján, hossz
  - Létrehozás például *Collection*-ból

XML formátum

```
<?xml version="1.0"?>
<employees>
  <person>
    <name>Big Joe</name>
    <address>Beach Street 12.</address>
    <phone>111-222</phone>
  </person>
  <person>
    <name>Small Joe</name>
    <address>Hill Street 13.</address>
    <phone>222-333</phone>
  </person>
</employees>
```

XML feldolgozás

- Az Android gazdag eszközkészletet biztosít XML-ek feldolgozására
- SAX alapú feldolgozás
  - *javax.xml.parsers.SAXParser*
  - Különféle függvényekkel dolgozhatjuk fel az értelmező által generált eseményeket
  - Az eseményeket akkor generálja az értelmező, amikor a jelölő nyelv meghatározott részeihez ér
- DOM alapú feldolgozás
  - *javax.xml.parsers.DocumentBuilder*
  - *javax.xml.parsers.DocumentBuilderFactory*
  - Memóriába kerül beolvasásra az XML mint egy „fa”
  - Lekérdezhetők az elemek

Külső osztálykönyvtárak XML és JSON feldolgozásra

- XML:
  - SimpleXML
- JSON:
  - GSON
- REST API tesztelésére:
  - Postman Chrome Client

GSON POJO példa (Java)

```java
public class PhoneInfo {
  @SerializedName("DeviceID")
  private String deviceId;
  @SerializedName("OperatingSystem")
  private String operatingSystem;

  public PhoneInfo(String deviceId, String operatingSystem) {
    this.deviceId = deviceId;
    this.operatingSystem = operatingSystem;
  }
  public String getDeviceId() {
    return deviceId;
  }
  public String getOperatingSystem() {
    return operatingSystem;
  }
}
```

GSON POJO példa (Kotlin)

```kotlin
class PhoneInfo(
        @SerializedName("DeviceID")
        val deviceId: String,

        @SerializedName("OperatingSystem")
        val operatingSystem: String
)
```

Aszinkron kommunikáció

UI módosítása más szálból

- Az alkalmazás indításakor a rendszer létrehoz egy úgynevezett *main* szálat (UI szál)
- Sokáig tartó műveletek blokkolhatják a felhasználói felületet, ezért új szálba kell indítani őket
- Az ilyen műveletek a végén az eredményt a UI-on jelenítik meg, azonban az Android a UI-t csak a fő szálból engedi módosítani!
- Több megoldás is szóba jöhet:
  - *Activity.runOnUiThread(Runnable)*
  - *View.post(Runnable)*
  - *View.postDelayed(Runnable, long)*
  - *Handler*
  - *AsyncTask és LocalBroadcast (példa: laboron szerepelni fog)*
  - *Külső libek, pl. EventBus, Otto*
  - *REST külső lib: RetroFit*

Aszinkron kommunikáció

UI módosítása más szálból

- Az alkalmazás indításakor a rendszer létrehoz egy úgynevezett *main* szálat (UI szál)
- Sokáig tartó műveletek blokkolhatják a felhasználói felületet, ezért új szálba kell indítani őket
- Az ilyen műveletek a végén az eredményt a UI-on jelenítik meg, azonban az Android a UI-t csak a fő szálból engedi módosítani!
- Több megoldás is szóba jöhet:
  - *Activity.runOnUiThread(Runnable)*
  - *View.post(Runnable)*
  - *View.postDelayed(Runnable, long)*
  - *Handler*
  - *AsyncTask és LocalBroadcast (példa: laboron szerepelni fog)*
  - *Külső libek, pl. EventBus, Otto*
  - *REST külső lib: RetroFit*

AsyncTask példa

```
class AsyncTaskUploadVote : AsyncTask<String, Void, String>() {

    override fun onPreExecute() {
        // ...
    }

    override fun doInBackground(vararg params: String): String? {
        val result = null
        // Hálózati kommunikáció, válasz mentése result-ba
        return result
    }

    override fun onPostExecute(result: String?) {
        // Eredmény használata UI szálon
        Log.d("RESULT", result)
    }

}

AsyncTaskUploadVote("Yes").execute()
```

REST API-k kezelése

Retrofit

- HTTP API megjelenítése Java interface formában

```
interface ItemsService {
    @GET("/items/{item}/details")
    fun listItems(@Path("item") item: String): Call<List<Item>>
}
```

- Retrofit osztály a konkrét implementáció generálására

```
val retrofit = Retrofit.Builder()
        .baseUrl("https://api.myshop.com")
        .build()
val service = retrofit.create(ItemsService::class.java)
```

- Mindenhívás az *ItemsService* mehet szinkron és aszinkron módon:

```
val items: Call<List<Item>> = service.listItems("myItem")
```

Retrofit

- HTTP kérések leírása annotációkkal:
  - URL és query paraméterek
  - Body – objektum konverzió (JSON, protocol buffers)
  - Multipart request és file feltöltés
- Gradle:
  - implementation 'com.squareup.retrofit2:retrofit:2.4.0'
- További információk:
  - http://square.github.io/retrofit/

Retrofit – GSON támogatás

- Automatikus konverzió a háttérben
  - Be kell állítani a Retrofitnak hogy mit használjon a konverzióhoz.

  - ```
    val retrofit=Retrofit.Builder()
            .baseUrl("http://api.myserver.com/")
            .addConverterFactory(GsonConverterFactory.create())
            .build()
    ```

- Gradle:
  - implementation 'com.google.code.gson:gson:2.8.5'
    implementation 'com.squareup.retrofit2:converter-gson:2.4.0'

- További információk:
  - http://square.github.io/retrofit/
    ```

Példa - Retrofit

- ISS API Astronauts
- Retrofit 2 + GSON

- http://api.open-notify.org/astros.json