Sip2Peer Android kliens

P2P alkalmazások a gyakorlatba

# Bevezetés

Az második mérés alkalmával az S2P csomag segítségével épített egyszerű gyűrűt. A gyűrű képes volt kezelni a hálózathoz csatlakozó újabb csomópontokat, és megvalósított egy naív útválasztó algoritmust is, így biztosítva, hogy a csomópontok képesek legyenek egymásnak üzeneteket eljuttatni. A csomópontok azonosítóikkal voltak címezhetőek, egyedül egy bootstrap node IP címét volt szükséges ismerni.

Az alkalmazáshoz egy egyszerű Java klienst is implementált, ami grafikus felületen keresztül tette elérhetővé a rendszer szolgáltatásait a felhasználók felé.

A harmadik mérés során a már elkészült hálózati magot egy Android alkalmazásba fogja beágyazni. Megismeri egy lehetséges architektúráját egy P2P Android alkalmazásnak. A mérés elvégezhető Android platformban való jártasság nélkül, de erősen ajánlott előzőleg megismerkedni a platform fontosabb koncepcióival.

# Minta alkalmazás felépítése

A minta-alkalmazás tartalmaz minden szükséges osztályt, a rendszerkomponensek be vannak regisztrálva az Android manifest file-ba, ami a szükséges jogosultságkérést is tartalmazza. Külső projektként hivatkozza az előző mérés során elkészített alkalmazást. Ez a hivatkozás, és a könyvtárakra való hivatkozás lehet hogy eltörik a projekt importálásakor.

## Alkalmazás komponensek

Az alkalmazás három Activity-ből és egy Service-ből áll. Az Activity-k egy egy felhasználói feladathoz kapcsolódnak:

* ***ConnectActivity***: Ennek a komponensnek a segítségével hozhat létre úgy csomópontot. A csomópont létrehozásakor döntheti el, hogy egy létező gyűrűhöz csatlakozik-e vagy új gyűrűt hoz létre.
* ***ContactListActivity:*** A peer által kezelt kapcsolatokat jeleníti meg, itt vehet fel új kapcsolatokat vagy kezdhet beszélgetést már ismert kapcsolatokkal.
* ***ConversatonActivity:*** Egy beszélgetést jelenít meg és lehetővé teszi új üzenetek küldését.
* **ChatService:** A RingPeer csomagolója, ez teszi elérhetővé a különböző alkalmazás-komponensek számára a csomópontot.

## Segédosztályok

Android platformon a lista-modellt a ListAdapter interface leszármaztatásával valósíthatja meg. A rendszer tartalmaz beépített lista-modelleket az egyszerű típusok kezelésére, összetett lista-elemek megjelenítéséhez azonban saját adaptert kell implementálni. Egy ilyen osztály a ChatMessageAdapter, mely egy két tagból álló lista-elemet valósít meg: a feladót és az üzenetet rendeli hozzá két szövegdobozhoz.

## Erőforrások

Az Android platform a megjelenítést és a program-logikát erőforrás-állományok segítségével különíti el. A megjelenítéssel kapcsolatos komponenseket lehetőség van programkódban dinamikusan létrehozni, vagy XML alapú erőforrásokból inflálni. Az erőforrás-kezelés megkönnyíti a többnyelvű alkalmazások vagy különböző képernyőméretű és arányú eszközökre optimalizált felületek implementálását, erről részletesebben az Android API [dokumentációjában](http://developer.android.com/guide/topics/resources/overview.html) olvashat.

A chat alkalmazás a string és layout típusú erőforrásokat használja. Előbbibe vannak kiszervezve a felületen megjelenítésre kerülő feliratok. Utóbbi a felhasználói felületet írja le.

A programkód és az erőforrások összekötésére az R osztály szolgál. Ezt a gen/R mappában találja. Az osztályt az SDK generálja az erőforrások alapján. Minden erőforráshoz, és a felületeken megjelenő komponensekhez konstans azonosítókat tartalmaz. A programkódból ezeken az azonosítókon keresztül érhetőek el az erőforrások.

A res/layout könyvtárban találja az egyes Activity-k felületét és a kontakt lista egy elemét ábrázoló felületet.

# Minta alkalmazás működése

Egy Java alkalmazásban a felület elemei szorosan köthetőek a backendhez, a példányosításon keresztül kontrollálhatjuk a komponensek felépítését, kölcsönös referenciákkal jól kézben tartható a komponensek kommunikációja. Ezzel szemben az Android világban managelt komponensekkel találkozunk. Az Android alkalmazások felépítése és életciklusa talán a webes megközelítéshez hasonló. Egy webes rendszerben a weboldalakat a böngésző "példányosítja" és JavaScript eseményeken keresztül biztosítja dinamikus elemek futtatását a betöltés és elkészülés különböző stádiumaiban. A felületen végzett események eredményeként a böngésző HTTP kéréseket intéz a back-endhez, ahol egy-egy kérés kiszolgálására a szerver hozza létre a megfelelő elemeket, amik valamilyen perzisztens tárra támaszkodva szolgálják ki azokat.

Képzelje el, hogy hogyan valósítana meg webkonténerben futó PHP-ben socket alapú P2P üzenetküldő rendszert, melynek kliensei böngészőben futnak!

Az Android platformon a böngészőben futó weboldalak megfelelői az Activity-k. Lehetőség van dinamikus felületek létrehozására, ahol a felhasználói interakció hatására programkódból épít fel komponensekből újabb felületeket, vagy a játékokban megismert módon canvas-t használ és pixelenként rajzolja meg az aktuális felhasználói felületet. De a leghatékonyabb, leggyorsabban implementálható megközelítésben minden felhasználói feladatot külön Activity lát el, mely statikus erőforrásokból építi fel felületét. Ahogyan webes környezetben a weboldalak közt átadásra kerülő adatokat request paraméterekké kell sorosítani, az Activity-k közti kommunikáció is sorosítva zajlik, a request szerepét az Intent veszi át, ennek lehet megadni a következő Activity osztályát, és a paramétereket kulcs-érték párok formájában. Ez maga után vonja, hogy nem lehet átadni referenciát: így nem fog tudni átadni egy inicializált RingPeer-t vagy egy megnyitott Socket-et.

Sok oka van annak, hogy ezt az életciklus modellt követi a rendszer, ezért csak különleges esetekben érdemes például egy Activity-be zsúfolni a teljes alkalmazást. Ezért az Activity-k fölött átívelő erőforrások kezelésére az Android platform három lehetőséget kínál:

* ***Singleton*** A Java-ból ismert design pattern, mely ClassLoader-enként egyetlen példányát engedélyezi egy osztálynak, mely az osztály statikus metódusán keresztül bárhonnan elérhető. Ez tekinthető az objektum-orientált világ politikailag korrekt globális változójának. Csak nagyon indokolt esetben használjuk, mivel nem managelt az életciklusa, így nincsenek jól bevált módszerek az általa foglalt erőforrások felszabadítására.
* ***Application*** Minden alkalmazáshoz tartozik egy Application osztály. Ez alapértelmezett eseben az Application, de az Android manifest-ben megadhatjuk az abból leszármazó saját osztályunkat is. Ebből minden időben csak egy példány létezik - így tekinthető managelt singleton-nak is - biztosan elérhető mindig, amikor az alkalmazás valamely komponense fut, a getApplication() metódus hívásán keresztül. Azonban az Application példányosítására, megszűnésére és újrapéldányosítására semmi ráhatásunk sincs, ezért használatát lehetőség szerint mellőzzük.
* ***Service*** Sejthető, hogy ez a javasolt megközelítés, a labor során is ezt fogja használni. A Service az Application-höz hasonlóan egyfajta managelt singleton. Ám ellentétben az Application-nel, példányosítására némi ráhatással bírunk: nem az alkalmazás indításával példányosul, hanem explicit kérésre. Kétféleképpen futtatható Service:
  + *Started service*: a service indítása után addig fut, ameddig explicite le nem állítják vagy erőforrások hiányában a rendszer fel nem szabadítja. Az egyes komponensek Intent-ekkkel kezdeményezhetik egy Service indítását. A már futó Service-ből nem indul új példány, de a futó példány megkapja az Intent-et, így a később beérkező igényeket is kezelni tudja, de erre a fejlesztőnek fel kell készülnie. A Service BroadcastIntent-ek segíségével jelezheti a kérés lefutását.
  + *Bound service*: a service-t az első csatlakozó komponens indítja, és addig fut, ameddig vannak csatlakozott komponensei. Ennek a megközelítésnek az az előnye, hogy a hívó komponens egy Binder-en keresztül referenciát szerezhet a Service-re, ami megkönnyíti a kommunikációt.

Jelen alkalmazásunknál a Service-nek folyamatosan futnia kell, ezért csak a started service megközelítés jöhet szóba: ha minden Acticity csatlakozna, majd megszűnésekor leiratkozna, akkor az Activity váltásoknál a Service megszűnne és új példány jönne belőle létre, elvesznének a megnyitott Socketek. Ugyanakkor a kétirányú kommunikáció megkönnyítésére a Binder előnyeit is ki fogja használni: egy elindított Service-hez ugyanúgy lehet csatlakozni, mint egy bound Service-hez.

A korábban elkészített projekt RingPeer osztályából fog egyet fenntartani a ChatService-ben, és ennek funkcionalitását fogja proxy-zni valamint kiterjeszteni hogy ki tudja szolgálni az alkalmazás igényeit.

# Feladatok

Importálja a letöltött projekteket!

Ha szükséges, javítsa ki az eltört referenciákat, hibás projekt tulajdonságokat!

Tekintse át a projekt szerkezetét, nézze meg a file-ok felépítését!

Ismerkedjen az Android specifikus Eclipse nézetekkel. Az erőforrás-szerkesztő segítségével tekintse át a felületeket!

Indítson el egy emulátort és indítsa el rajta az alkalmazást!

Az idő rövidsége és az esetlegesen hiányzó Android tapasztalatok miatt a mérés során nem fog felhasználói felületet fejleszteni, és az eseménykezelők inicializálását is tartalmazza a projekt. A feladata a Service elkészítése lesz.

Nyissa meg a ChatService osztályt, és valósítsa meg a TODO üzenetekben elhelyezett feladatokat.

Nézze meg, hogy hol és hogyan használják az Activity-k a Service-t.

Próbálja ki az alkalmazást emulátoron majd valódi eszközön.