Servisi
WorkManager

prof. dr Bratislav Predić
dipl. inž. Nevena Tufegdžić

Razvoj mobilnih aplikacija i servisa
Računarstvo i informatika

2023.

Android Servis

- Servis u Andoidu je komponenta aplikacije koja može da izvršava operacije u pozadini
- Servis ne poseduje korisnički interfejs – nema implementacije Composable funkcija
- Servisi se često koriste za I/O sa fajlovima, izvršavanje mrežnih procesa ili operacije kao što je puštanje muzike

Android Servis

- Servis ne kreira zasebnu nit sam po sebi, već se izvršava u glavnoj niti
- Ukoliko je potrebno da se servis izvršava iz druge niti, to se mora obezbediti korišćenjem korutina
- Poželjno je izvršavati funkcionalnosti servisa u zasebnoj niti ukoliko on sadrži blokirajuće operacije, kako se sama aplikacija ne bi blokirala

Android Servis

- Servis nije ni proces ni nit!
- Servis je način da aplikacija naznači sistemu šta radi u pozadini, čak i kada korisnik ne interaguje direktno sa aplikacijom
- Sa druge strane, servis je takođe i način da aplikacija pruži neku svoju funkcionalnost drugim aplikacijama

Tipovi servisa

- Servisi u Androidu mogu biti:
  - **Foreground** – servisi koji izvršavaju nešto što korisnik vidi
  - **Background** – servisi koji rade nešto što korisnik ne vidi
  - **Bound** – servisi koji nude „klijent-server“ interfejs, tako da druge komponente mogu da koriste njihove funkcionalnosti

Foreground servisi

- Foreground servisi izvršavaju operacije koje su vidljive korisniku
- Servis se izvršava čak i kada korisnik aktivno ne koristi aplikaciju
- Iz tih razloga, potrebno je prikazati notifikaciju koja stoji sve dok je servis aktivan (npr. aplikacija za puštanje muzike)
- Notifikacija nestaje tek kada se servis terminira ili izmesti iz foreground-a

Background servisi

- Background servisi se izvršavaju u pozadini i njihove operacije nisu vidljive korisniku (npr. mrežni servisi)
- Nije potrebno prikazivati notifikaciju jer servis ne zahteva komunikaciju sa korisnikom

Bound servisi

- Bound servisi su aktivni samo dok ih makar jedna komponenta neke aplikacije koristi
- Komponente se *binduju* na servis i na taj način koriste njegove funkcionalnosti
- Više komponenti se može vezati na jedan Bound servis u isto vreme, ali kada se poslednja *unbinduje*, servis prestaje sa radom
- Servis može u isto vreme da bude i Bound i startovan

Kreiranje servisa

- Svaki servis se mora deklarisati u AndroidManifest.xml fajlu:

```
<manifest ... >
  ...
  <application ... >
      <service android:name=".HelloService" />
      ...
  </application>
</manifest>
```

Kreiranje servisa

- Svaki servis nasleđuje Service klasu, čije metode je potrebno override-ovati

Service	
onStartCommand()	Poziva se kada se startuje servis
onBind()	Poziva se kada se binduje servis
onCreate()	Poziva se kada se kreira servis
onDestroy()	Poziva se kada se uništi servis

Metode servisa

- **onStartCommand()** je metoda koja se poziva kada neka komponenta startuje servis pomoću startService()
- Kada se izvrši onStartCommand(), servis je aktivan i biće aktivan sve dok ga neka komponenta ne ugasi metodom stopService(), ili dok ne ugasi sam sebe pozivom stopSelf()
- Ako servis treba samo da bude samo Bound service, nije potrebno implementirati ovu metodu

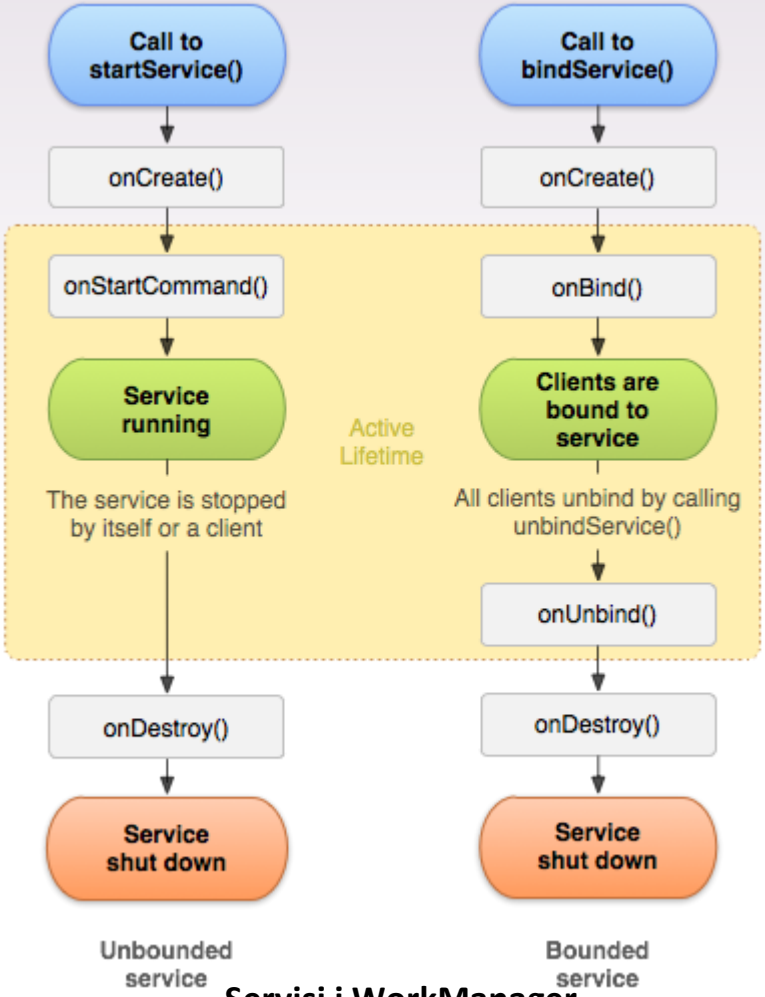
Metode servisa

- **onBind()** se poziva kada neka komponenta želi da se binduje na servis pozivom metode bindService()
- Ova metoda se mora implementirati
- Povratna vrednost ove metode mora biti objekat koji implementira IBinder interfejs
- Ako servis ne treba da bude Bound service, onda metoda samo treba da vraća null

Metode servisa

- **onCreate()** se poziva kada sistem kreira servis
  - Poziva se pre onStartCommand() i onBind()
- **onDestroy()** se poziva kada sistem uništava servis
  - Može se pozvati kada se sve komponente unbinduju sa servisa, ili nakon poziva stopService() ili stopSelf()
  - Ovo je poslednja metoda koja može da se pozove u servisu

Životni ciklus servisa



Životni ciklus servisa

- Android sistem stopira servis kada mu fale sistemski resursi za Activity koji je u fokusu korisnika
- Servis koji koristi Activity ima manje šanse da bude stopiran
- Ako je servis deklarisan kao Foreground, ima manje šanse da bude stopiran
- Što je servis duže u *running* fazi, to su mu veće šanse da bude stopiran

```
class HelloService : Service() {

    private var serviceLooper: Looper? = null
    private var serviceHandler: ServiceHandler? = null

    // Handler that receives messages from the thread
    private inner class ServiceHandler(looper: Looper) : Handler(looper) {

        override fun handleMessage(msg: Message) {
            // Do some work
            try {
                Thread.sleep(5000)
            } catch (e: InterruptedException) {
                // Restore interrupt status.
                Thread.currentThread().interrupt()
            }

            // Stop the service using the startId, so that we don't stop
            // the service in the middle of handling another job
            stopSelf(msg.arg1)
        }
    }
```

```
override fun onCreate() {
        // Start up the thread running the service.  Note that we create a
        // separate thread because the service normally runs in the process's
        // main thread, which we don't want to block.  We also make it
        // background priority so CPU-intensive work will not disrupt our UI.
        HandlerThread("ServiceStartArguments", Process.THREAD_PRIORITY_BACKGROUND).apply {
            start()

            // Get the HandlerThread's Looper and use it for our Handler
            serviceLooper = looper
            serviceHandler = ServiceHandler(looper)
        }
}
```

```
override fun onStartCommand(intent: Intent, flags: Int, startId: Int): Int {
    Toast.makeText(this, "service starting", Toast.LENGTH_SHORT).show()

    // For each start request, send a message to start a job and deliver the
    // start ID so we know which request we're stopping when we finish the job
    serviceHandler?.obtainMessage()?.also { msg ->
        msg.arg1 = startId
        serviceHandler?.sendMessage(msg)
    }

    // If we get killed, after returning from here, restart
    return START_STICKY
}
```

Povratna vrednost onStartCommand

- onStartCommand() metoda treba da vrati celobrojnu vrednost
- Ova povratna vrednost opisuje način na koji Android sistem treba da obradi situaciju kada mora da stopira (*kill*) servis
- Moguće konstante (definisane u Android Framework-u) su:
  - START_NON_STICKY
  - START_STICKY
  - START_REDELIVER_INTENT

Povratna vrednost onStartCommand

- START_NON_STICKY
  - Ako sistem ubije servis nakon izvršenja onStartCommand(), servis se neće ponovo kreirati
  - Servis se kreira ponovo samo u slučaju da postoje PendingIntent-ovi koje treba da primi
  - Ova vrednost se koristi kako servis ne bi radio onda kada nije potrebno i kad sve komponente koje koriste servis mogu da ga ponovo pokrenu same ukoliko je potrebno

Povratna vrednost onStartCommand

- START_STICKY
  - Ako sistem ubije servis nakon izvršenja onStartCommand(), servis se ponovo kreira i ponovo se poziva onStartCommand()
  - Bitno: sistem ne prosleđuje ponovo Intent pomoću kog se kreira servis!
  - Ukoliko ne postoje PendingIntent-i koje servis treba da obradi, servis dobija null vrednost kao Intent

Povratna vrednost onStartCommand

- START_REDELIVER_INTENT
  - Ako sistem ubije servis nakon izvršenja onStartCommand(), servis se ponovo kreira i ponovo se poziva onStartCommand()
  - Razlika između START_STICKY i START_REDELIVER_INTENT: servis se startuje tako što mu se prosleđuje poslednji primljeni Intent
  - PendingIntent-ovi se takođe prosleđuju

```
override fun onBind(intent: Intent): IBinder? {
    return null
}

override fun onDestroy() {
    Toast.makeText(this, "service done", Toast.LENGTH_SHORT).show()
}
```

Startovanje servisa

- Servis se može startovati iz Activity-ja ili neke druge aplikacije tako što se kreira Intent i prosleđuje startService() ili startForegroundService() metodi

```
val serviceIntent = Intent(this, HelloService::class.java)
startService(serviceIntent)
```

Startovanje servisa

- Pri pozivu startService() ili startForegroundService(), sistem poziva onStartCommand() metodu servisa
- Ukoliko servis nije već kreiran, sistem prvo poziva onCreate(), pa zatim onStartCommand()
- Prosleđuje mu se Intent iz startService() ili startForegroundService()

Startovanje Bound servisa

- Ovaj servis treba da implementira onBind() metodu koja treba da vraća IBinder
- IBinder definiše interfejs pomoću kog se komunicira sa servisom
- Klijentska komponenta koristi ovaj interfejs kako bi komunicirala sa servisom

```
class LocalService : Service() {
    // Binder given to clients.
    private val binder = LocalBinder()

    // Random number generator.
    private val mGenerator = Random()

    /** Method for clients.  */
    val randomNumber: Int
        get() = mGenerator.nextInt(100)

    /**
     * Class used for the client Binder. Because we know this service always
     * runs in the same process as its clients, we don't need to deal with IPC.
     */
    inner class LocalBinder : Binder() {
        // Return this instance of LocalService so clients can call public methods.
        fun getService(): LocalService = this@LocalService
    }

    override fun onBind(intent: Intent): IBinder {
        return binder
    }
}
```

Stopiranje Bound servisa

- Nakon što klijentska komponenta završi sa radom sa servisom, treba da pozove unbindService() metod
- Više klijentskih komponenti sme da se binduje na isti servis istovremeno
- Onda kada se sve komponente unbinduju, servis se uništava
- Nije potrebno eksplicitno pozivati stopService()

WorkManager

- Deo Android Jetpack biblioteke
- Pruža API za raspoređivanje task-ova koje treba izvršiti u pozadini
- Omogućava izvršavanje task-ova čak i kada aplikacija nije startovana
- Koristi se kao optimizacija za akcije koje troše bateriju (primena filtera na sliku, pribavljanje podataka sa servera, itd)

WorkManager

- Izvršava task samo kada su uslovi za njegovo pokretanje ispunjeni
- Podržava ponovno pokretanje (retry) ili odlaganje (reschedule) task-ova
- Može da pokrene task čak i ako se uređaj u međuvremenu restartuje

WorkManager

- Potrebno je dodati dependency u build.gradle fajl:

```
dependencies {
    // WorkManager dependency
    implementation "androidx.work:work-runtime-ktx:2.8.1"
}
```

WorkManager - implementacija

- Rad sa WorkManager-om odvija se pomoću tri glavne klase:
  - Worker
  - WorkRequest
  - WorkManager

WorkManager - implementacija

- **Worker** je klasa u kojoj se nalazi kod koji predstavlja ono što task izvršava
- Mora da nasledi klasu Worker i da implementira njen doWork() metod
- WorkManager koristi doWork() metodu kako bi izvršavao task u pozadini i asinhrono

WorkManager - implementacija

```
class BlurWorker (context: Context, workerParams: WorkerParameters)
    : Worker(context, workerParams) {

    override fun doWork() : Result {
        // do work here
    }

}
```

WorkManager - implementacija

- **WorkRequest** je zahtev koji se prosleđuje WorkManager-u kako bi izvršio neki task
- Pomoću WorkRequest-a se mogu podesiti ograničenja pod kojima se task pokreće
- WorkRequest sadrži i tip Worker-a koji treba da izvrši dati task

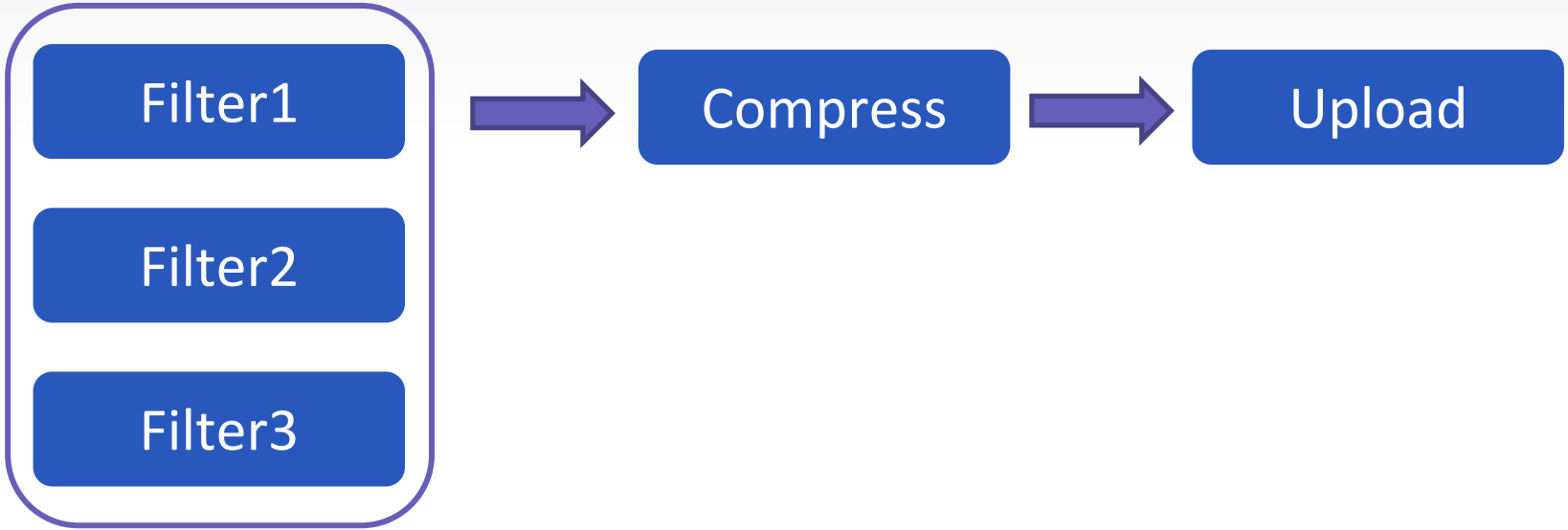
WorkManager - implementacija

```
val chargingConstraint = Constraints.Builder()
    .setRequiresCharging(true)
    .build()

val blurWorkRequest = OneTimeWorkRequestBuilder<BlurWorker>()
    .setConstraints(chargingConstraint)
    .build()

workManager.enqueue(blurWorkRequest)
```

Nadovezivanje task-ova

- WorkManager podržava i nadovezivanje taskova
- Ovo se koristi kada je potrebno:
  - Izvršiti task-ove u paraleli
  - Sačekati da se jedan task završi kako bi sledeći iskoristio njegove rezultate

Nadovezivanje task-ova



Nadovezivanje task-ova

```
WorkManager.getInstance(context)
    .beginWith(listOf(filterWorker1, filterWorker2, filterWorker3))
    .then(compressWorker)
    .then(uploadWorker)
    .enqueue()
```

Rezime

- Servisi pružaju mogućnost izvršenja neke funkcionalnosti u okviru aplikacije van glavne niti, i pružanje funkcionalnosti drugim aplikacijama
- Foreground, Background, Bound
- Nasleđuju Service klasu
- Nemaju UI
- Moraju se deklarisati u AndroidManifest.xml fajlu

Rezime

- Foreground servisi izvršavaju akcije koje su vidljive korisniku, i zato treba da prikazuju notifikaciju dokle god su aktivni
- Background servisi ne izvršavaju akcije koje su vidljive korisniku
- Bound servisi omogućavaju pristup funkcionalnostima aplikacije drugim aplikacijama
- Servis nije proces i servis nije nit, ali može i poželjno je da svoje funkcije izvršava u posebnim nitima

Rezime

- Foreground servis se startuje metodom startForegroundService()
- Background servis se startuje pozivom startService()
- Bound servis se startuje kada se neka komponenta bind-uje na servis, pozivom bindService()
- Ovim metodama se prosleđuje eksplicitni Intent koji startuje servis

Rezime

- WorkManager je „clean“ način izvršavanja task-ova u pozadini bez blokiranja aplikacije ili trošenja resursa (pogotovo baterije)
- Svaki task modelovan je klasom koja nasleđuje Worker klasu i implementira njenu doWork() metodu, koja mora da vraća Result
- WorkManager izvršava task koji mu je prosleđen u okviru WorkRequest-a, pomoću kog se takođe mogu specificirati i uslovi za pokretanje datog task-a

Literatura

- Services overview | Android Developers
- https://developer.android.com/courses/pathways/android-basics-kotlin-unit-6-pathway-1
- https://developer.android.com/topic/libraries/architecture/workmanager/

Hvala na pažnji!