Servisi i prijemnici poruka

Mobilne aplikacije

Stevan Gostojić

Fakultet tehničkih nauka, Novi Sad

8. novembar 2022.

Pregled sadržaja

Proces

- aktivnost (angažovanje procesora na izvršavanju programa)
- slika (deo operativne memorije koji sadrži naredbe u mašinskom jeziku i podatke na stack-u i heap-u)
- atributi (stanje, prioritet, itd.)

Nit

- Nit je redosled izvršavanja naredbi u procesu
- Jedan proces može da sadrži više niti (onda svaka nit sadrži stack, stanje i prioritet i izvršava relativno nezavisnu sekvencu naredbi)

Raspoređivanje niti

- Različite niti mogu da se izvršavaju na jednom procesoru (konkurentno) ili na više procesora (paralelno)
- Kako jedan procesor ne može istovremeno da izvršava više niti, one se moraju izvršavati naizmenično
- S obzirom da različite niti mogu da pristupaju istom resursu, potrebno je voditi računa o sinhronizaciji niti

Razlika između procesa i niti

- Niti se koriste za "male" zadatke, a procesi za "velike" zadatke (izvršavanje aplikacije)
- Niti koje pripadaju istom procesu dele isti adresni prostor (to znači da mogu da komuniciraju direktno preko operativne memorije)
- Procesi ne dele isti adresni prostor (to znači da je komunikacija između procesa složenija i sporija od komunikacije između niti)

Android i procesi

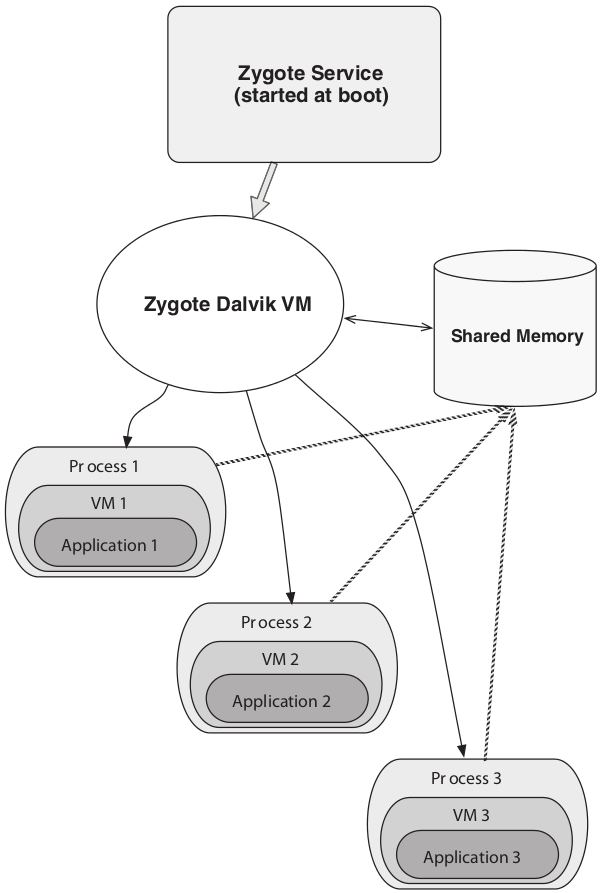

Figure 1: Android i procesi.

- Kada Android startuje prvu komponentu neke aplikacije, startuje je u novom procesu sa jednom niti
- Svaka sledeća komponenta iste aplikacije startuje se u istom procesu i u istoj niti kao i prva komponenta
- Moguće je startovati različite komponente iste aplikacije u različitim procesima ili različite komponente različitih aplikacija u istom procesu (mada to nije preporučljivo)

Android i procesi

- Android zadržava procese u operativnoj memoriji što je duže moguće
- Da bi se slobodila memorija za procese višeg prioriteta, nekada je potrebno "ubiti" proces nižeg prioriteta
- Prioritet procesa se određuje na osnovu vrste i stanja komponenti koje sadrži kao i prioriteta drugih procesa koji od njega zavise
- Zato bi aktivnosti i prijemnici poruka koji izvršavaju dugačke operacije trebalo da startuju servis umesto niti

Prioritet procesa

- `foreground` (proces sadrži aktivnost koja se nalazi u prvom planu, servis koji je vezan za aktivnost koja se nalazi u prvom planu, servis koji se izvršava u prvom planu, servis koji izvršava jednu od metoda životnog ciklusa ili prijemnik poruka koji izvršava onReceive metodu)
- `visible` (proces sadrži vidljivu aktivnost ili servis koji je vezan za vidljivu aktivnost)
- `service` (proces sadrži servis)
- `background` (proces sadrži aktivnost koja se nalazi u pozadini)
- `empty` (proces ne sadrži komponente)

Android i niti

- Android izvršava aplikaciju (tj. njene komponente) u glavnoj niti
- Ova nit je, između ostalog, zadužena za slanje i primanje poruka od komponenti korisničkog interfejsa (zato se zove i UI nit)
- Stoga nije preporučljivo blokirati UI nit ("application isn't responding" dijalog) i pristupati komponentama korisničkog interfejsa iz drugih niti (nisu thread-safe)
- Metode životnog ciklusa servisa i dobavljača sadržaja moraju biti thread-safe

Android i niti

```
1 // Pogresno (blokiranje UI niti)
2 public void onClick(View v) {
3   Bitmap b = loadImageFromNetwork(imageUrl);
4   imageView.setImageBitmap(b);
5 }
6
```

Android i niti

```
1  // Pogresno (GUI komponente nisu thread-safe)
2  public void onClick(View v) {
3    new Thread(new Runnable() {
4      public void run() {
5        Bitmap b = loadImageFromNetwork(imageUrl);
6        imageView.setImageBitmap(b);
7      }
8      }).start();
9  }
10
```

Android i niti

```
1  // Ispravo
2  public void onClick(View v) {
3    new Thread(new Runnable() {
4      public void run() {
5        Bitmap b = loadImageFromNetwork(imageUrl);
6        imageView.post(new Runnable() {
7          public void run() {
8            imageView.setImageBitmap(b);
9          }
10       });
11     }
12   }).start();
13 }
14
```

Pregled sadržaja

Rukovaoci

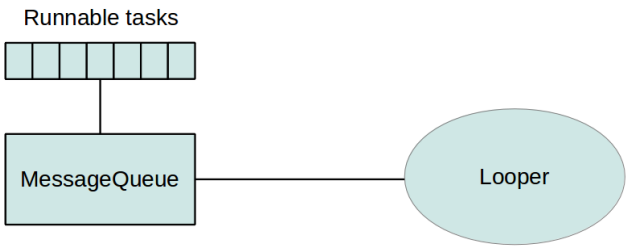

Figure 2: Rad sa nitima.

- Red poruka (`MessageQueue`) je red koji sadrži poruke koje je potrebno obraditi (zadatke koje je potrebno izvršiti)
- Rukovaoc (`Handler`) obrađuje poruke (izvršava zadatke) koje se nalaze u redu poruka
- Looper održava nit "u životu" i prosleđuje poruke (zadatke) iz reda poruka rukovaocu na obradu

Rukovaoci

```
1  class LooperThread extends Thread {
2    public Handler handler;
3
4    public void run() {
5      Looper.prepare();
6
7      handler = new Handler() {
8        public void handleMessage(Message msg) {
9          // process incoming messages here
10         // this will run in non-ui/background thread
11       }
12     };
13
14     Looper.loop();
15   }
16 }
17
```

Rukovaoci

```
1 Message message = new Message();
2 message.obj = "Ali send message";
3 handler.sendMessage(message);
4
```

Rukovaoci

```
new Handler(Looper.getMainLooper()).post(
  new Runnable() {
    @Override
    public void run() {
      // this will run in the main thread
    }
});

```

Rukovaoci

Za obradu poruka potrebno je implementirati `void handleMessage(Message msg)` i pozvati:

- `boolean sendEmptyMessage(int)`
- `boolean sendMessage(Message)`
- `boolean sendMessageAtTime(Message, long)`
- `boolean sendMessageDelayed(Message, long)`

Rukovaoci

Za izvršavanje proizvoljnog koda potrebno je pozvati:

- `boolean post(Runnable)`
- `boolean postAtTime(Runnable, long)`
- `boolean postDelayed(Runnable, long)`

Rukovaoci

```
1  // Reuses existing handler
2  handler = getWindow().getDecorView().getHandler();
3
4  // Uses UI component's post() method
5  ImageView imageView = (ImageView) findViewById(R.id.image_view);
6  imageView.post(new Runnable() {
7    @Override
8    public void run() {
9      imageView.setImageBitmap(bitmap);
10   }
11 });
12
```

Pregled sadržaja

Asinhroni zadatak

- Asinhrnoni zadatak (`AsyncTask`) olakšava asinhrono izvršavanje operacija
- Automatski izvršava blokirajuću operaciju u pozadinskoj niti, vraća rezultat UI niti i nudi dodatne funkcije (kao što je obaveštavanje o progresu operacije)
- Svi asinhroni zadaci jedne aplikacije izvršavaju se u jednoj niti (oni se serijalizuju)

Asinhroni zadatak

```
1 public void onClick(View v) {
2   new DownloadImageTask().execute(imageUrl);
3 }
4
```

Asinhroni zadatak

```
1  private class DownloadImageTask extends AsyncTask<String, Void,
       Bitmap> {
2    // The system calls this to perform work in a worker
3    // thread and delivers it the parameters given to
4    // execute()
5    protected Bitmap doInBackground(String... urls) {
6      return loadImageFromNetwork(urls[0]);
7    }
8
9    // The system calls this to perform work in the UI
10   // thread and delivers the result from doInBackground()
11   protected void onPostExecute(Bitmap result) {
12     imageView.setImageBitmap(result);
13   }
14 }
15
```

Asinhroni zadatak

`AsyncTask` je generička klasa koja koristi tri tipa:

- `params` (tip parametara koji se prosleđuju pozadinskoj niti)
- `progress` (tip jedinice u kojoj se meri progres operacije)
- `result` (tip povratne vrednosti koju vraća pozadinska nit)

Asinhroni zadatak

Sadrži četiri generičke metode:

- `void onPreExecute()` - poziva se u UI niti pre izvršavanja zadatka
- `Result doInBackground(Params... params)` - poziva se u pozadinskoj niti odmah posle onPreExecute
- `void onProgressUpdate(Progress... values)` - poziva se u UI niti posle poziva publishProgress u pozadinskoj niti
- `void onPostExecute(Result result)` - poziva se u UI niti posle izvršavanja zadatka

Pregled sadržaja

Servisi

- Servis je komponenta koja izvršava "duge" operacije u pozadini (startovan servis) i služi za implemantaciju klijent-server arhitekture (vezan servis)
- Servis se izvršava u istoj niti u kojoj se izvršavala komponenta koja ga je startovala (čak i ako ta komponenta više nije aktivna)
- Druga komponenta može da se veže za servis i da sa njime komunicira (i ako se nalazi u drugom procesu)

Servisi

- Servis može biti startovan ili vezan (može istovremeno biti i startovan i vezan, ali se to retko koristi)
- Startovan servis se izvršava neodređeno vreme (servis treba da se sam zaustavi kada izvrši operaciju)
- Vezan servis se izvršava samo dok je neka komponenta vezana za njega (nudi interfejs koji omogućava komponentama da komuniciraju sa njim šaljući zahteve i dobijajući odgovore)

Životni ciklus servisa

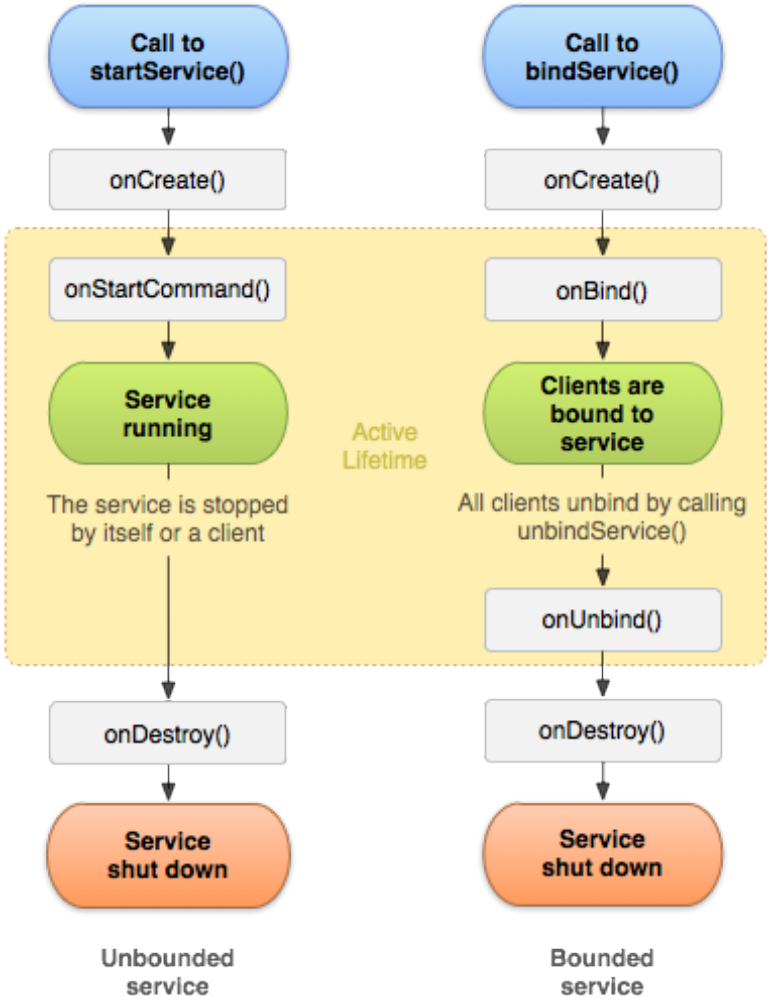

Figure 3: Životni ciklus servisa.

Životni ciklus servisa

Servisi, poput aktivnosti, sadrže metode koje se pozivaju prilikom prelaska iz jednog u drugo stanje:

- `onCreate` (poziva se prilikom stvaranja servisa)
- `onStartCommand` (poziva se posle poziva startService metode)
- `onBind` (poziva se posle poziva bindService metode)
- `onUnbind` (poziva se posle poziva unbindService metode)
- `onRebind` (poziva se posle poziva bindService ako je prethodno izvršena onUnbind metoda)
- `onDestroy` (poziva se prilikom uništavanja servisa)

Životni ciklus servisa

- Razlikuje se ceo životni vek servisa (između poziva `onCreate` i `onDestroy` metoda) i
- aktivni životni vek (počinje pozivom `onStartCommand` ili `onBind` metode, a završava se pozivom `onDestroy` ili `onUnbind` metode)

Pravljenje servisa

Servis može da se napravi nasleđivanjem klasa:

- `Service` (u ovom slučaju je važno startovati pozadinsku nit u kojoj će se izvršiti operacije i voditi računa u sinhronizaciji ukoliko više komponenti istovremeno koriste isti servis)
- `IntentService` (u ovom slučaju će se operacije automatski izvršiti u pozadinskoj niti i pozivi metoda servisa će se automatski sinhronizovati)

AndroidManifest.xml

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest ... >
3   <application ... >
4     <service android:name=".ExampleService" />
5   </application>
6 </manifest>
7
```

ExampleService.java

```
1  public class ExampleService extends Service {
2    @Override
3    public void onCreate() {
4      // ...
5    }
6
7    @Override
8    public int onStartCommand(Intent intent, int flags,
       int startId) {
9      // ...
10
11     stopSelfResult(startId);
12
13     // If we get killed, after returning from here,
       restart
14     return START_NOT_STICKY;
15   }
16
17   @Override
18   public IBinder onBind(Intent intent) {
19     // We don't provide binding, so return null
20     return null;
21   }
22
23   public void onDestroy() {
24     // ...
25   }
26 }
27
```

Servisi

Constant	Meaning
`START_NOT_STICKY`	If the system kills the service after `onStartCommand()` returns, do not recreate the service.
`START_STICKY`	If the system kills the service after `onStartCommand()` returns, recreate the service and call `onStartCommand()`.
`START_REDELIVER_INTENT`	If the system kills the service after `onStartCommand()` returns, recreate the service and call `onStartCommand()` with the last intent delivered.

Table 1: Vrednosti `flags` parametra.

ExampleService.java

```
1  public class ExampleService extends IntentService {
2    // A constructor is required, and must call the super
3    // IntentService(String) constructor with a name for
4    // the worker thread
5    public ExampleService() {
6      super("ExampleService");
7    }
8
9    // The IntentService calls this method from the
       default
10   // worker thread with the intent that started the
       service.
11   // When this method returns, IntentService stops the
12   // service, as appropriate.
13   @Override
14   protected void onHandleIntent(Intent intent) {
15     // ...
16   }
17 }
18
```

Pokretanje servisa

```
1 Intent intent = new Intent(this, ExampleService.class);
2 startService(intent);
3
```

Zaustavljanje servisa

- Servis se može zaustaviti sam pozivom stopSelf metode, može za zaustaviti druga komponenta pozivom stopService metode ili ga može zaustaviti Android platforma (da bi oslobodila memoriju)
- Aplikacije bi trebalo da zaustave svoje servise čim izvrše operaciju da se ne bi trošili resursi (npr. baterija)

Pokretanje servisa u prvom planu

- Servis se može pokrenuti u prvom planu pozivom startForeground metode, a ukloniti iz prvog plana pozivom stopForeground metode
- Trebalo bi da se nalazi u prvom planu ukoliko je korisnik svestan servisa (što znači da ne treba da se "ubije" u nedostatku memorije)
- Servis u prvom planu mora obezbediti obaveštenje u statusnoj liniji

ExampleService.java

```
1  public class ExampleService extends Service {
2    @Override
3    public int onStartCommand(Intent intent, int flags, int startId) {
4      // ...
5      Notification notification = ...;
6      startForeground(ONGOING_NOTIFICATION_ID, notification);
7      // ...
8      stopForeground(true);
9      // ...
10 }
11
```

Bound Service

- Prilikom pravljenja servisa za koji mogu da se vežu druge komponente, mora se napraviti i interfejs koji omogućava klijentima da komuniciraju sa servisom.
- To se može uraditi na tri načina:
  - nasleđivanjem `Binder` klase (ako se servis i klijent izvršavaju u istom procesu)
  - korišćenjem `Messenger` klase (ako se servis i klijent ne izvršavaju nužno u istom procesu)
  - korišćenjem AIDL (isto kao i u prethodnom slučaju)

ExampleService.java

```
1  public class ExampleService extends Service {
2    // Binder given to clients
3    private IBinder binder = new ExampleBinder();
4
5    // Class used for the client Binder
6    public class ExampleBinder extends Binder {
7      ExampleService getService() {
8        // Return this instance of ExampleService
9        return ExampleService.this;
10     }
11   }
12
13   @Override
14   public IBinder onBind(Intent intent) {
15     return binder;
16   }
17
```

ExampleService.java

```
1     // Random number generator
2     private Random generator = new Random();
3
4     // Method for clients
5     public int getRandomNumber() {
6       return generator.nextInt(100);
7     }
8   }
9
```

ExampleActivity.java

```
1  public class ExampleActivity extends Activity {
2    private LocalService service;
3
4    private boolean bound = false;
5
6    // Defines callbacks for service binding, passed to
       bindService()
7    private ServiceConnection connection = new
       ServiceConnection() {
8      @Override
9      public void onServiceConnected(ComponentName cn,
       IBinder s) {
10       ExampleBinder binder = (ExampleBinder) s;
11       service = binder.getService();
12       bound = true;
13     }
14
15     @Override
16     public void onServiceDisconnected(ComponentName arg0
       ) {
17       bound = false;
18     }
19   };
20
```

ExampleActivity.java

```
1   @Override
2   protected void onCreate(Bundle bundle) {
3     super.onCreate(bundle);
4     setContentView(R.layout.main);
5   }
6
7   @Override
8   protected void onStart() {
9     super.onStart();
10    Intent intent = new Intent(this, ExampleService.
      class);
11    bindService(intent, connection, Context.
      BIND_AUTO_CREATE);
12  }
13
14  @Override
15  protected void onStop() {
16    super.onStop();
17    if (bound) {
18      unbindService(connection);
19    }
20  }
21
```

ExampleActivity.java

```
1  // Called when a button is clicked
2  public void onButtonClick(View v) {
3    if (bound) {
4      // Call a method from the ExampleService.
5      int num = service.getRandomNumber();
6      Toast.makeText(this, "number: " + num, Toast.LENGTH_SHORT).show
       ();
7    }
8  }
9
```

Servisi

Constant	Meaning
`BIND_AUTO_CREATE`	Automatically create the service as long as the binding exists.
`BIND_DEBUG_UNBIND`	Include debugging help for mismatched calls to unbind.
`BIND_IMPORTANT`	This service should be brought to the foreground level.
`BIND_NOT_FOREGROUND`	Don't allow this binding to raise the target service's process to the foreground level.

Table 2: Vrednosti `flags` parametra.

Pregled sadržaja

Prijemnici poruka

- Prijemnici poruka (`BroadcastReceiver`) obrađuju događaje (opisane objektima klase Intent)
- Te događaje može da izazove Android platforma ili druga komponenta (koja može da se nalazi u drugoj aplikaciji)

AndroidManifest.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest ... >
3    <application ... >
4      <receiver android:name=".SMSReceiver">
5        <intent-filter>
6          <action name="android.provider.Telephony.SMS_RECEIVED
       " />
7        </intent-filter>
8      </receiver>
9    </application>
10 </manifest>
11
```

SMSReceiver.java

```
1 public class SMSReceiver extends BroadcastReceiver {
2   @Override
3     public void onReceive(Context context, Intent intent
    ) {
4       // ...
5   }
6 }
7
```

Prijemnici poruka

Parametri `onReceive` metode su:

- `context` (kontekst u kome se izvršava onReceive metoda)
- `intent` (namera koja opisuje događaj koji treba obraditi)

Namera prosleđena `startActivity` ili `startService` metodi neće prouzrokovati događaj koji će obraditi `onReceive` metoda (važi i obrnuto)

Prijemnici poruka

Konstanta	Značenje
`ACTION_BATTERY_LOW`	A warning that the battery is low.
`ACTION_HEADSET_PLUG`	A headset has been plugged into the device, or unplugged from it.
`ACTION_SCREEN_ON`	The screen has been turned on.
`ACTION_SHUTDOWN`	Device is shutting down.

Table 3: Neke od akcija (događaja) koje se mogu obraditi.

Prijemnici poruka

- Metoda `onReceive` se poziva iz glavne niti (to znači da "dugačke" operacije treba izvršavati u posebnom servisu koji startuje posebnu nit)
- Prijemnik poruka postoji samo u toku izvršavanja `onReceive` metode (zato se u ovoj metodi ne mogu izvršiti asinhrone operacije kao što su prikazivanje dijaloga ili vezivanje za servis)
- Proces u kome se izvršava `onReceive` metoda ima foreground prioritet (nakon toga, prioritet procesa određuju ostale komponente koje se u njemu nalaze)

Prijemnici poruka

Postoje dve vrste događaja koje prijemnici poruka mogu da obrade:

- normalni događaji (asinhroni su, prijemnici ih obrađuju nedefinisanim redosledom i njigova obrada je efikasnija)
- uređeni događaji (obrađuje ih više prijemnika redom, a svaki prijemnik može da prosledi događaj sledećem prijemniku ili da potpuno obustavi njegovu obradu)

ExampleActivity.java

```
1 public class ExampleActivity extends Activity {
2   @Override
3   public void onCreate(Bundle bundle) {
4     Intent intent = new Intent();
5     intent.setAction(SMS_RECEIVED);
6     sendBroadcast(intent);
7   }
8 }
9
```

SMSReceiver.java

```
1 public class SMSReceiver extends BroadcastReceiver {
2   @Override
3   public void onReceive(Context context, Intent intent)
      {
4     // ...
5   }
6 }
7
```

ExampleActivity.java

```
1 public class ExampleActivity extends Activity {
2   @Override
3   public void onCreate(Bundle bundle) {
4     Intent intent = new Intent();
5     intent.setAction(SMS_RECEIVED);
6     sendOrderedBroadcast(intent, null);
7   }
8 }
9
```

AndroidManifest.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest ... >
3    <application ... >
4      <receiver android:name=".SMSReceiver">
5        <intent-filter android:priority="100">
6          <action name="android.provider.Telephony.SMS_RECEIVED" />
7        </intent-filter>
8      </receiver>
9    </application>
10 </manifest>
11
```

SMSReceiver.java

```
1 public class SMSReceiver extends BroadcastReceiver {
2   @Override
3   public void onReceive(Context context, Intent intent)
      {
4     if (condition) abortBroadcast() {
5       // ...
6     }
7   }
8 }
9
```

Prijemnici poruka

Metoda	Značenje
`abortBroadcast()`	Sets the flag indicating that this receiver should abort the current broadcast.
`getResultCode()`	Retrieve the current result code, as set by the previous receiver.
`getResultData()`	Retrieve the current result data, as set by the previous receiver.
`setResultCode(int code)`	Change the current result code of this broadcast.
`setResultData(String data)`	Change the current result data of this broadcast.

Table 4: Metode koje se koriste za spregu između prijemnika poruka.

Prijemnici poruka

- Prijemnici poruka omogućavaju razmenu poruka između komponenti različitih aplikacija
- To znači da treba obratiti pažnju na moguće zloupotrebe
- Moguće je primeniti prava pristupa prilikom slanja poruke (prosleđujući permission parametar `sendBroadcast` metodi) ili prilikom prijema poruke (postavljajući permission atribut receiver elementa)
- Komponenta koja salje/prima poruku mora imati odgovarajuće pravo pristupa (zatraženo `<uses-permission>` elementom u `AndroidManifest.xml`)

Agenda

Zakazivanje

- Tajmer (`Timer`) zakazuje izvršavaje jednokratnih zadataka (u apsolutnom trenutku ili posle relativnog kašnjenja) ili zadataka koji se ponavljaju (sa fiksim periodom ili fiksnom frekvencijom)
- Svaki tajmer ima jednu nit koja zadatke izvršava sekvencijalno (to znači da može da dođe do kašnjenja u izvršavanju zadatka ukoliko je ta nit zauzeta)
- Komponenta koja je zakazala izvršavanje zadatka ne mora biti aktivna u trenutku u kome zadatak treba da se izvrši (to znači da zadatak može da se ne izvrši)

Zakazivanje

Metoda	Značenje
`schedule(TimerTask task, Date when)`	Schedule a task for repeated fixed-delay execution after a specific time has been reached.
`schedule(TimerTask task, long delay)`	Schedule a task for single execution after a specified delay.
`cancel()`	Cancels the Timer and all scheduled tasks.

Table 5: Metode klase `Timer`.

Zakazivanje

Metoda	Značenje
`schedule(TimerTask task, Date when, long period)`	Schedule a task for repeated fixed-delay execution after a specific time has been reached.
`schedule(TimerTask task, long delay, long period)`	Schedule a task for repeated fixed-delay execution after a specific delay.
`scheduleAtFixedRate (TimerTask task, long delay, long period)`	Schedule a task for repeated fixed-rate execution after a specific delay has passed.
`scheduleAtFixedRate (TimerTask task, Date when, long period)`	Schedule a task for repeated fixed-rate execution after a specific time has been reached.

Table 6: Metode klase texttttTimer.

SplashScreen.java

```
1  public class SplashScreen extends Activity {
2
3    public static final int SPLASH_TIMEOUT = 1500;
4
5    @Override
6    protected void onCreate(Bundle bundle) {
7      super.onCreate(bundle);
8      setContentView(R.layout.splash_screen);
9      new Timer().schedule(new TimerTask() {
10         @Override
11         public void run() {
12           startActivity(new Intent(SplashScreen.this, MyMovies.class)
       );
13           finish();
14         }
15       }, SPLASH_TIMEOUT);
16   }
17 }
18
```

Zakazivanje

- Klasa `AlarmManager` omogućuje pristup sistemskom alarmu i startovanje aplikacije u nekom trenutku u budućnosti
- Kada se alarm aktivira, sistem emituje objekat klase `Intent`, što kao posledicu ima automatsko startovanje aplikacije (ukoliko već nije startovana)

AndroidManifest.xml

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest ... >
3    <application ... >
4
5      <receiver android:name=".PortfolioStartupReceiver">
6        <intent-filter>
7          <action android:name="android.intent.action.
       BOOT_COMPLETED" />
8        </intent-filter>
9      </receiver>
10
11     <receiver android:name=".AlarmReceiver">
12       <!-- -->
13     </receiver>
14
15     <service android:name=".PortfolioManagerService">
16       <!-- -->
17     </service>
18
19   </application>
20 </manifest>
21
```

PortfolioStartupReceiver.java

```
1  public class PortfolioStartupReceiver extends BroadcastReceiver {
2
3    private static final int FIFTEEN_MINUTES = 15*60*1000;
4
5    @Override
6    public void onReceive(Context context, Intent intent) {
7      AlarmManager mgr = (AlarmManager)context.getSystemService(Context.
       ALARM_SERVICE);
8      Intent i = new Intent(context, AlarmReceiver.class);
9      PendingIntent sender = PendingIntent.getBroadcast(
10       context, 0, i, PendingIntent.FLAG_CANCEL_CURRENT);
11     Calendar now = Calendar.getInstance();
12     now.add(Calendar.MINUTE, 2);
13     mgr.setRepeating(
14       AlarmManager.RTC_WAKEUP, now.getTimeInMillis(),FIFTEEN_MINUTES, sender);
15   }
16 }
17
```

AlarmReceiver.java

```
1 public class AlarmReceiver extends BroadcastReceiver {
2   @Override
3   public void onReceive(Context context, Intent intent) {
4     Intent stockService = new Intent(context, PortfolioManagerService.class);
5     context.startService(stockService);
6   }
7 }
8
```

PortfolioManagerService.java

```
1 public class PortfolioManagerService extends Service {
2   @Override
3   public int onStartCommand(Intent intent, int flags, int startId) {
4     updateStockData();
5     return Service.START_NOT_STICKY;
6   }
7 }
8
```