Prijemnici poruka

Mobilne aplikacije

Pregled sadržaja

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

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
  <uses-permission android:name="android.permission.RECEIVE_SMS"/>
  <!-- dynamic permission request should also be made -->

  <application ... >
    <receiver android:name=".SMSReceiver">
      <intent-filter>
        <action name="android.provider.Telephony.SMS_RECEIVED" />
      </intent-filter>
    </receiver>
  </application>
</manifest>

```

SMSReceiver.java

```
public class SMSReceiver extends BroadcastReceiver {
  @Override
    public void onReceive(Context context, Intent intent) {
      // ...
  }
}
```

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_POWER_CONNECTED`	External power has been connected to the device.
`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 1: 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

- Informacije o događajima se do prijemnika poruka prenose putem namera.
- Kao što namere mogu biti eksplicitne i implicitne tako se i broadcast poruke mogu podeliti na eksplicitne (namenjene konkretnom prijemniku poruka) i implicitne (prijemnik poruka se pronalazi na osnovu filtera namera).

Registrovanje prijemnika poruka

- Prijemnik poruka je moguće registrovati statički (putem manifest fajla) ili dinamički (iz programkog koda).
- Primer statički registrovanog prijemnika poruka:

```
<receiver android:name=".SMSReceiver">
  <intent-filter>
    <action name="android.provider.Telephony.SMS_RECEIVED" />
  </intent-filter>
</receiver>
```

Registrovanje prijemnika poruka

- Primer dinamički registrovanog prijemnika poruka:

```
MyBroadcastReceiver myBroadcastReceiver = new MyBroadcastReceiver();
IntentFilter filter = new IntentFilter("android.provider.Telephony.
    SMS_RECEIVED");
registerReceiver(myBroadcastReceiver, filter);

```

- Pri tome je prijemnik poruka implementiran na uobičajen način:

```
MyBroadcastReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
      // ...
    }
}

```

Prijemnici poruka

- Sistemski događaji koriste implicitne namere kako bi različite komponente mogle primiti poruke o tim događajima.
- Prijavljivanje prijemnika poruka na određenu vrstu implicitno definisanih poruka može dovesti do preopterećenja resursa uređaja ukoliko postoji veći broj prijemnika poruka koji očekuju istu vrstu poruke.
- Tako je za neke implicitne broadcast poruke koje generiše sistem dovoljno koristiti <intent-filter>. To su one poruke koje se ne dešavaju često ili se ređe javlja potreba za obradom te vrste događaja. Primeri takvih broadcast poruka su opisani akcijama: ACTION_BOOT_COMPLETED, ACTION_TIMEZONE_CHANGED, ACTION_USB_DEVICE_ATTACHED, ACTION_USB_DEVICE_DETACHED, ACTION_MEDIA_MOUNTED, ACTION_MEDIA_UNMOUNTED.

Prijemnici poruka

- Za ostale sistemske događaje je potrebno koristi dinamičku registraciju prijemnika poruka. Na primer, to je slučaj sa događajima opisanim akcijom: CONNECTIVITY_ACTION
- Pored dinamičke registracije prijemnika poruka, može biti neophodan i dinamički (runtime) zahtev za odgovarajućim permisijama potrebnim za obradu te vrste događaja.
- Osim toga, za neke sistemske događaje u novijim verzijama Androida se ne emituju broadcast poruke. To je slučaj sa: ACTION_NEW_PICTURE, ACTION_NEW_VIDEO. Tada se preporučuje korišćenje JobScheduler klase.

Prijemnici poruka

Postoje dva vrste događaja koje prijemnici poruka mogu obraditi:

- normalni događaji (asinhroni su, prijemnici ih obrađuju nedefinisanim redosledom i njihova 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

```
public class ExampleActivity extends Activity {
  @Override
  public void onCreate(Bundle bundle) {
    Intent intent = new Intent();
    intent.setAction(SMS_RECEIVED);
    sendBroadcast(intent);
  }
}
```

SMSReceiver.java

```
public class SMSReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    // ...
  }
}
```

ExampleActivity.java

```
public class ExampleActivity extends Activity {
  @Override
  public void onCreate(Bundle bundle) {
    Intent intent = new Intent();
    intent.setAction(SMS_RECEIVED);
    sendOrderedBroadcast(intent, null);
  }
}


// 2nd parameter is null because no special permissions
    are needed

```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
  <application ... >
    <receiver android:name=".SMSReceiver">
      <intent-filter android:priority="100">
        <action name="android.provider.Telephony.SMS_RECEIVED" />
      </intent-filter>
    </receiver>
  </application>
</manifest>
```

SMSReceiver.java

```
public class SMSReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    // ...
    if (condition) {
        abortBroadcast();
    }
  }
}
```

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 2: 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`)

Prijemnici poruka i permisije - prijem poruke

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ...>

    <!-- We define custom permission -->
    <permission android:name="my.app.PERMISSION" />
  <!-- Additional protection can be set using the protection level:
   android:protectionLevel="signature" will require the sender app to be signed
     with the same certificate,
   android:protectionLevel="dangerous" will asks user to grant this permission
    -->

    <!-- The receiver demands this permission -->
    <receiver
        android:name="my.app.BroadcastReceiver"
        android:permission="my.app.PERMISSION">
        <intent-filter>
            <action android:name="my.app.Action" />
        </intent-filter>
    </receiver>
     ...
</manifest>
```

Prijemnici poruka i permisije - slanje poruke

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ...>

    <!-- we want to use our custom permission -->
    <uses-permission android:name="my.app.PERMISSION" />
    <!--  now this app is allowed to send broadcasts
          to the application that declared this permission -->
    <!--  depending on the protection level it may be necessary
          to sign the app or the user to grant this permission -->
     ...
</manifest>

```

Agenda

Zakazivanje

- Tajmer (`Timer`) zakazuje izvršavanje jednokratnih zadataka (u apsolutnom trenutku ili posle relativnog kašnjenja) ili zadataka koji se ponavljaju (sa fiksnim 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 single execution when 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 3: Metode klase `Timer`.

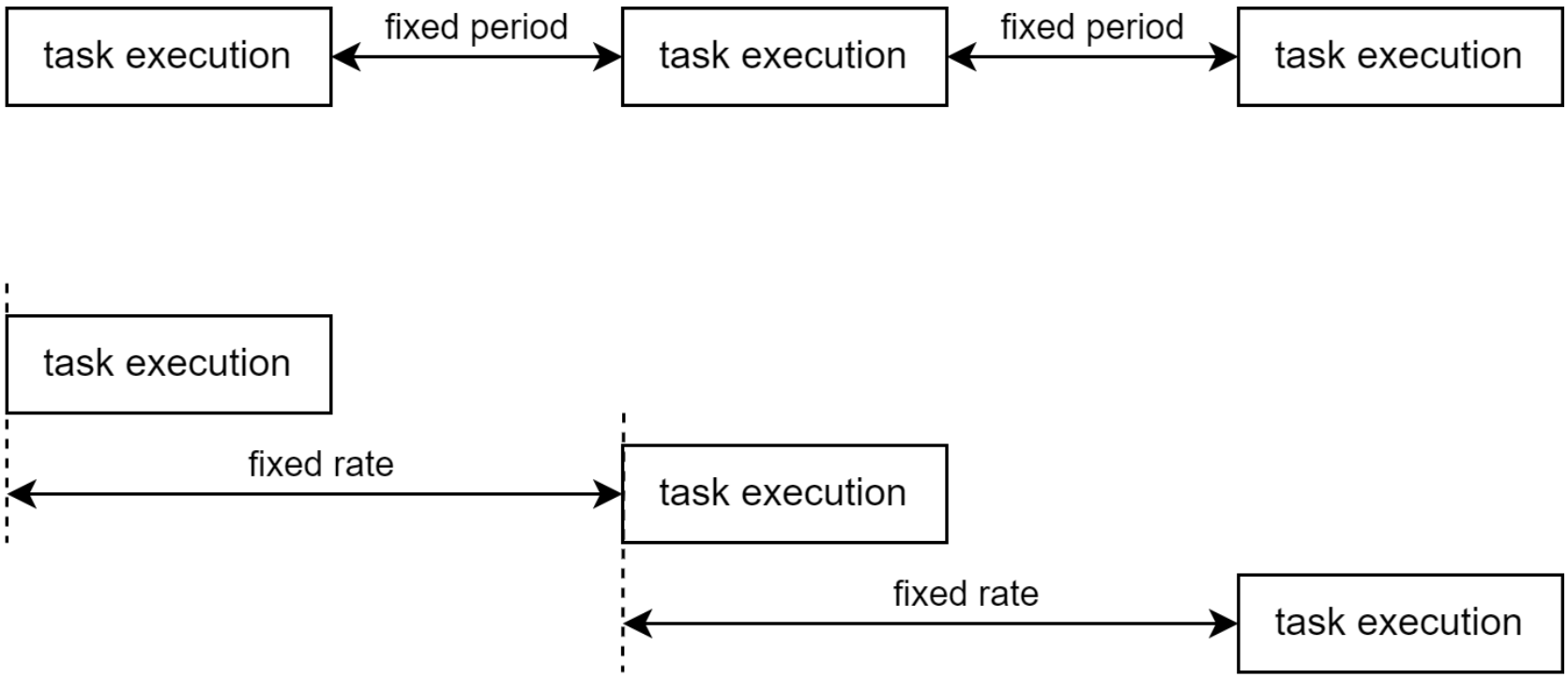
Zakazivanje

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 4: Metode klase `Timer`.

schedule vs. scheduleAtFixedRate

- izvršavanje taskova pomoću schedule metode (gornji primer) i scheduleAtFixedRate metode (donji primer)



SplashScreen.java

```
public class SplashScreen extends Activity {

  public static final int SPLASH_TIMEOUT = 1500;

  @Override
  protected void onCreate(Bundle bundle) {
    super.onCreate(bundle);
    setContentView(R.layout.splash_screen);
    new Timer().schedule(new TimerTask() {
        @Override
        public void run() {
          startActivity(new Intent(SplashScreen.this, MyMovies.class));
          finish();
        }
      }, SPLASH_TIMEOUT);
  }
}

```

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

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
  <application ... >

    <receiver android:name=".PortfolioStartupReceiver">
      <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED" />
      </intent-filter>
    </receiver>

    <receiver android:name=".AlarmReceiver">
      <!-- -->
    </receiver>

    <service android:name=".PortfolioManagerService">
      <!-- -->
    </service>

  </application>
</manifest>
```

PortfolioStartupReceiver.java

```
public class PortfolioStartupReceiver extends BroadcastReceiver {

  private static final int FIFTEEN_MINUTES = 15*60*1000;

  @Override
  public void onReceive(Context context, Intent intent) {
    AlarmManager mgr = (AlarmManager)context.getSystemService(Context.ALARM_SERVICE);
    Intent i = new Intent(context, AlarmReceiver.class);
    PendingIntent pi = PendingIntent.getBroadcast(
        context, 0, i, PendingIntent.FLAG_CANCEL_CURRENT);
    Calendar now = Calendar.getInstance();
    now.add(Calendar.MINUTE, 2);
    mgr.setRepeating(
        AlarmManager.RTC_WAKEUP, now.getTimeInMillis(),FIFTEEN_MINUTES, pi);
  }
}
```

AlarmReceiver.java

```
public class AlarmReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    Intent stockService = new Intent(context, PortfolioManagerService.class);
    context.startService(stockService);
  }
}
```

PortfolioManagerService.java

```
public class PortfolioManagerService extends Service {
  @Override
  public int onStartCommand(Intent intent, int flags, int startId) {
    updateStockData();
    return Service.START_NOT_STICKY;
  }
}

```