Servisi i prijemnici poruka

Fakultet tehničkih nauka, Novi Sad

Pregled sadržaja

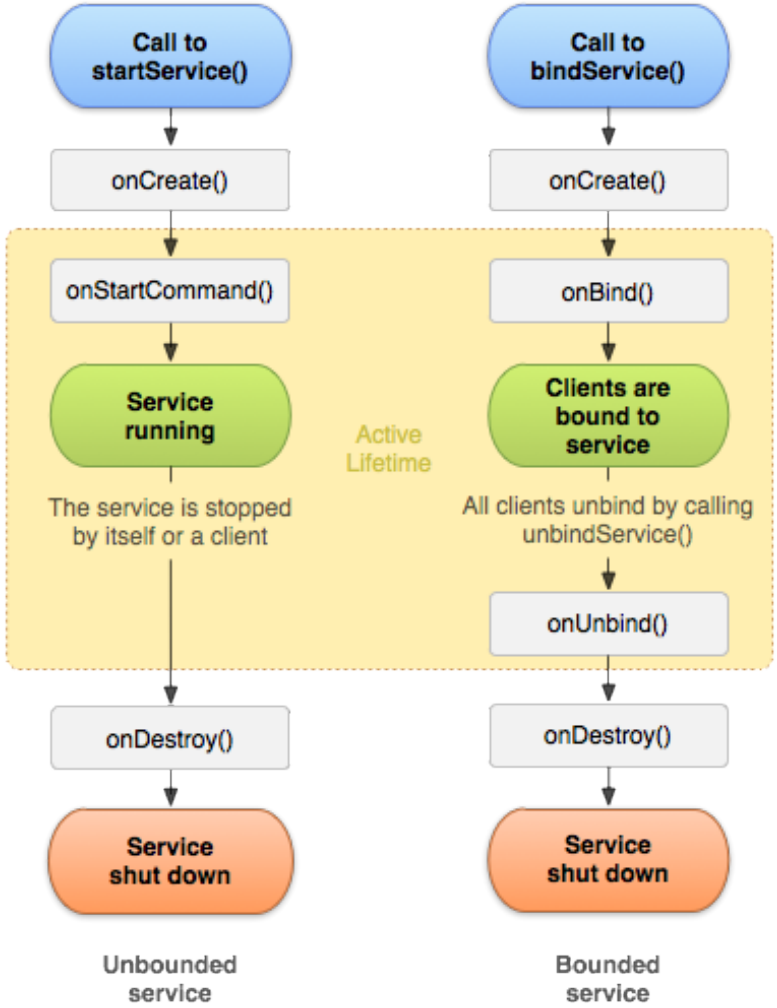
Servisi

- Servis je komponenta koja izvršava "duge" operacije u pozadini i služi za implemantaciju klijent-server arhitekture
- 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



Slika: 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)

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>
```

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 }
```

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.

Tabela: 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 }
```

Pokretanje servisa

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

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

```java
public class ExampleService extends Service {
  @Override
  public int onStartCommand(Intent intent, int flags, int startId) {
    // ...
    Notification notification = ...;
    startForeground(ONGOING_NOTIFICATION_ID, notification);
    // ...
    stopForeground(true);
    // ...
}
```

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   }
```

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  }
```

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   };
```

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    }
```

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     }
```

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.

Tabela: Vrednosti `flags` parametra.