

ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
Dipartimento di Informatica – Scienza e Ingegneria (DISI)
C.d.S. in Ingegneria e Scienze Informatiche, Campus di Cesena

Programmazione in Android

Service

Angelo Croatti
a.croatti@unibo.it

Sistemi Embedded e Internet of Things
A.A. 2019 – 2020

Outline

- 1 Overview
 - Started Service vs. Bound Service
 - Service Lifecycle
- 2 Gestione degli Started Service
 - Intent Service
- 3 Gestione dei Bound Service
 - Uso dei Binder
 - Uso dei Messenger



Service (Overview)

- Rappresenta il componente dedicato all'esecuzione di compiti long-running in background.
 - ▶ Senza interazione con l'utente (non è prevista UI per i Service).
- Un Service può continuare la sua esecuzione in background anche se l'applicazione che l'ha avviato viene interrotta.
- Oltre ad eseguire i propri compiti in background, un Service può offrire un'interfaccia mediante la quale richiedere l'esecuzione di compiti al Service da altri componenti (tipicamente activity).
- Esistono, essenzialmente, due *tipi* di Service:
 - ▶ Started Service
 - ▶ Bound Service

Started Service

- Avviato da un altro componente, tramite un Intent passato come parametro al metodo `startService(Intent i)` definito nella classe `android.app.Activity`.
- Può restare in esecuzione in background per un tempo indefinito.
- Generalmente, esegue un preciso compito e non restituisce nessun risultato al componente che lo ha creato.
- Al termine dell'esecuzione del proprio compito, termina se stesso chiamando il metodo `stopSelf()` definito nella classe `android.app.Service`.

Bound Service

- Avviato da un altro componente, tramite un Intent passato come parametro alla funzione `bindService(Intent i, ServiceConnection s, int flags)` definito nella classe `android.content.Context`.
- Offre un'interfaccia Client-Server che consente ad altri componenti di interagire con il Service.
 - ▶ Per richiedere l'esecuzione di compiti, ottenere risultati,...
- Il suo tempo di vita è condizionato al fatto che ci sia almeno un altro componenti collegato (bound) al Service.
 - ▶ Più componenti possono essere collegati al Service contemporaneamente ma quando l'ultimo si scollega, il servizio viene distrutto dal sistema.

Service (note)

1. Nonostante la suddivisione tra servizi *Started* e *Bound*, uno stesso Service può lavorare contemporaneamente in entrambi le modalità.
2. **Ciascun Service viene eseguito nel Main Thread.**
 - ▶ Ovvero, non crea un thread separato né viene eseguito in un processo a se stante.
 - ▶ Pertanto, se il Service deve eseguire computazione CPU-intensive o compiti bloccanti, questi devono essere demandati ad un thread separato creato opportunamente all'interno del Service.
3. Come le Activity, anche i Service devono essere dichiarati nel File Manifest per poter essere identificati dal sistema.

Esempio

```
<application>
  <service android:name="com.example.MyService"
    android:label="@string/service_name"/>
</application>
```

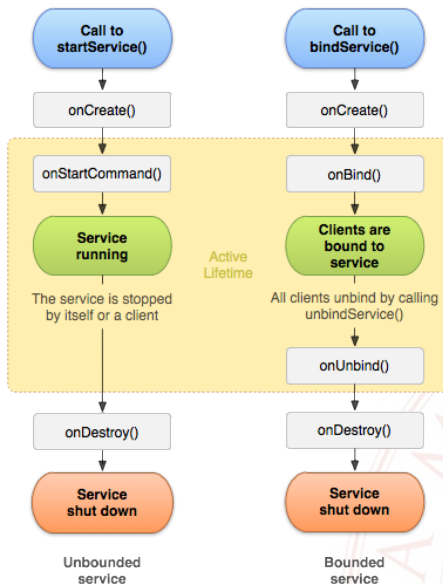
Service o Thread?

- Spesso può sorgere il dubbio relativo al fatto di utilizzare un service (started) o un thread vero e proprio per eseguire un compito.
- Vale la seguente regola:
 - ▶ Se si ha la necessità di continuare a svolgere il compito anche quando l'utente non sta interagendo con l'applicazione allora deve essere utilizzato un Service.
 - ▶ Viceversa, se deve essere eseguito un compito rilevante solo mentre l'utente sta interagendo con l'applicazione allora è preferibile creare ed eseguire un thread da un'activity dell'applicazione (eventualmente tramite un Async Task).

Service Lifecycle I

- Analogamente ad un'Activity, anche il componente Service prevede un proprio lifecycle.
 - ▶ Anche in questo caso, alle transizioni di stato sono associate alcune callback che possono essere ridefinite.
- Per i Service, non è necessario chiamare la versione originale della superclasse nella ridefinizione della callback.
- Nel caso di Started Service, devono essere ridefiniti i metodi `onCreate()`, `onStartCommand()` e `onDestroy()`.
- Nel caso di Bound Service, devono essere ridefiniti i metodi `onCreate()`, `onBind()`, `onUnbind()` e `onDestroy()`.

Service Lifecycle II



Service Lifecycle – Callback I

```
void onCreate()
```

- Invocata successivamente alla creazione del Service.

```
int onStartCommand(Intent i, int flags, int startId)
```

- Invocata dopo la creazione del Service, qualora la creazione sia stata richiesta attraverso la funzione `startService()`.

```
void onDestroy()
```

- Invocata prima che il sistema distrugga il Service non più utilizzato.

Service Lifecycle – Callback II

`IBinder onBind(Intent i)`

- Invocata dopo il collegamento di un componente al Service attraverso la funzione `bindService()`.
- Restituisce un'istanza di un oggetto di tipo `IBinder` che consente al chiamante di interagire con il Service.

`boolean onUnbind(Intent i)`

- Invocata dopo che un componente precedentemente collegato al Service, abbia deciso di scollegarsi attraverso la funzione `unbindService()`.

Outline

- 1 Overview
 - Started Service vs. Bound Service
 - Service Lifecycle
- 2 Gestione degli Started Service
 - Intent Service
- 3 Gestione dei Bound Service
 - Uso dei Binder
 - Uso dei Messenger



Creazione e Avvio di uno Started Service

- Deve essere creata un'istanza di un oggetto che estenda dalla classe `android.app.Service`.
- Il componente che crea il servizio deve predisporre un apposito Intent ed invocare il metodo `startService(Intent i)` per avviare il Service.

Esempio

```
Intent i = new Intent(this, MyService.class);  
startService(i);
```

- L'invocazione del metodo `startService()` dal componente che crea il Servizio, provoca l'invocazione del metodo `onStartCommand()` sul Servizio, dopo l'invocazione del metodo `onCreate()` qualora il Service non sia mai stato creato in precedenza.

Intent Service

- Creare uno Started Service definendo un oggetto che estende direttamente dalla classe `Service` pone una serie di complicazioni.
 - ▶ Tra le altre, devono essere gestiti esplicitamente i thread per l'esecuzione dei compiti del service.
- Alternativamente, può essere estesa la classe `android.app.IntentService` che a sua volta estende da `Service`.
 - ▶ Semplifica notevolmente la creazione di uno Started Service.
 - ▶ Deve essere definito un costruttore che chiami il costruttore padre, passando un nome per il worker thread che sarà associato automaticamente al Service.
 - ▶ Deve essere ridefinito il metodo `onHandleIntent(Intent i)` che viene invocato quando il servizio viene avviato. Inoltre, quando il metodo termina, contestualmente termina anche il servizio.

Intent Service - Esempio

Esempio

```
public class MyService extends IntentService{

    public MyService() {
        super("MyServiceWorkerThread");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        //do something
    }

}
```

NOTA: Qualora si decida di ridefinire anche le altre callback, in questo caso bisogna assicurarsi che vengano invocate esplicitamente le medesime callback della superclasse.

Outline

- 1 Overview
 - Started Service vs. Bound Service
 - Service Lifecycle
- 2 Gestione degli Started Service
 - Intent Service
- 3 Gestione dei Bound Service
 - Uso dei Binder
 - Uso dei Messenger



Creazione di un Bound Service

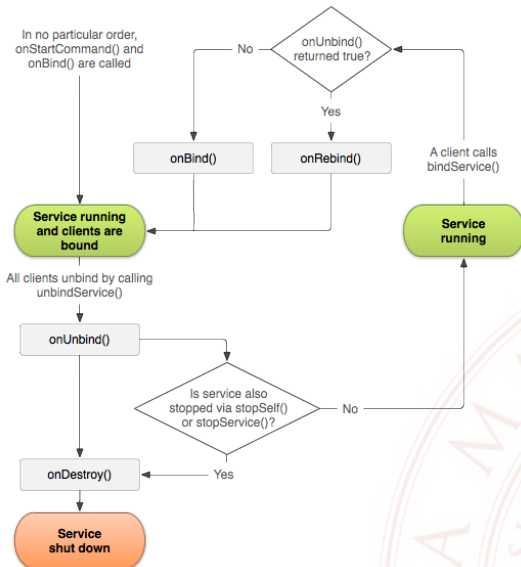
- Deve essere creata un'istanza di un oggetto che estenda dalla classe `android.app.Service`.
- Il componente che crea il servizio deve predisporre un apposito Intent ed invocare il metodo `bindService(Intent i, ServiceConnection sc, int flags)` per avviare il Service.

Esempio

```
ServiceConnection conn = ... ;  
  
Intent i = new Intent(this, MyBoundService.class);  
bindService(i, conn, Context.BIND_AUTO_CREATE);
```

- Deve essere predisposto un opportuno canale di comunicazione (`binder`) che consenta di interagire con il servizio.

Bound Service Lifecycle (dettagli)



Bound Service con Binder – Esempio I

Descrizione: *Si vuole creare un servizio che fornisca un numero casuale ai componenti collegati a fronte di specifiche richieste.*

Esempio – Definizione del Servizio

```
public class RndGenService extends Service{
    private final IBinder binder = new RndGenBinder();
    private final Random generator = new Random();

    @Override
    public IBinder onBind(Intent intent){
        return binder;
    }

    /* RandomGenService API methods */
    public int getRandomNumber() {
        return generator.nextInt(100);
    }
}
```

Bound Service con Binder – Esempio II

```
/* Binder for the RndGenService */  
class RndGenBinder extends Binder {  
    public RndGenService getService() {return RndGenService.this;}  
}
```

- Il metodo `onBind()` deve restituire l'istanza di un oggetto di tipo **IBinder** opportunamente definito come inner-class del servizio.
 - ▶ Tale oggetto (`RandomGenBinder`, nell'esempio) deve esporre un metodo pubblico (`getService()`) che restituisca l'istanza del servizio a cui fa capo il binder.
- Devono essere poi predisposti tutti i metodi che possono essere invocati sul servizio per chiedere l'esecuzione di compiti ("API" del servizio).

Bound Service con Binder – Esempio III

Esempio – Activity che usa il Servizio

```
public class MyActivity extends Activity{
    private RndGenService service;
    private boolean bounded = false;

    private ServiceConnection conn = new ServiceConnection(){
        @Override
        public void onServiceConnected(ComponentName cls, IBinder bnd){
            service = ((RndGenService.RndGenBinder) bnd).getService();
            bounded = true;
        }

        @Override
        public void onServiceDisconnected(ComponentName cls){
            bounded = false;
        }
    };
};
```

Bound Service con Binder – Esempio IV

```
@Override
protected void onCreate(Bundle savedInstanceState){/*...*/}

@Override
protected void onStart(){
    super.onStart();

    Intent intent = new Intent(this, RndGenService.class);
    bindService(intent, conn, Context.BIND_AUTO_CREATE);
}

@Override
protected void onStop(){
    super.onStop();

    if(bounded){
        unbindService(conn);
        bounded = false;
    }
}
```

Bound Service con Binder – Esempio V

```
    }  
}  
  
public void onClick(View v){  
    if(bounded){  
        int nextRndNum = service.getRandomNumber();  
    }  
}
```

- Il binding al servizio è possibile attraverso la definizione di un oggetto di tipo **android.content.ServiceConnection** che stabilisce e mantiene la connessione tra l'activity e il servizio.

Bound Service con Binder – Esempio VI

- Per tale oggetto, devono essere definiti i metodi `onServiceConnected()` e `onServiceDisconnected()`.
 - ▶ Il primo dei due consente di recuperare il riferimento al servizio, avvalendosi del Binder (definito come inner class del servizio) e del relativo metodo `getService()`.
 - ▶ Tramite tale riferimento sarà poi possibile invocare i metodi resi pubblici dal Service.
- L'oggetto che rappresenta la connessione è passato, insieme all'Intent per la creazione del servizio, al metodo `bindService()`.
- Il servizio può essere rilasciato chiamando il metodo `unbindService()`, passando come parametro il riferimento alla connessione.

Bound Service con Binder – Note

- L'uso esplicito di un Binder per la connessione al servizio è consentito solo quando il componente che vuole avvalersi del servizio appartiene alla stessa applicazione.
- Viceversa, deve essere usato il meccanismo basato su **Messenger**, che consente di richiedere l'esecuzione di metodi definiti in servizi di altri processi/applicazioni.
 - ▶ Ovvero, quando non è possibile avere il riferimento esplicito all'oggetto istanza del Service.

Bound Service con Messenger – Esempio I

Esempio – Definizione del Servizio di Messaging

```
public class MessengerService extends Service {  
    static final int MESSAGE_1_REQUEST = 1;  
    static final int MESSAGE_2_REQUEST = 2;  
  
    final Messenger m = new Messenger(new IncomingHandler());  
  
    @Override  
    public IBinder onBind(Intent intent){  
        return m.getBinder();  
    }  
  
    private class IncomingHandler extends Handler{  
        @Override  
        public void handleMessage(Message msg){  
            switch(msg.what){  
                case MESSAGE_1_REQUEST:  
                    //do something  
            }  
        }  
    }  
}
```

Bound Service con Messenger – Esempio II

```
        break;
    case MESSAGE_2_REQUEST:
        //do something
        break;
    default:
        super.handleMessage(msg);
        break;
    }
}
}
```

- Il servizio definisce come campo interno un oggetto di tipo **android.os.Messenger**
 - ▶ Inizializzato mediante un Handler progettato in modo specifico per ricevere i messaggi dai componenti chiamanti.

Bound Service con Messenger – Esempio III

- Il metodo `onBind()` restituisce il riferimento al binder ottenuto direttamente dall'istanza del Messenger.

Esempio – Activity (esterna) che usa il servizio

```
public class ActivityMessenger extends Activity{
    private Messenger service = null;
    private boolean bounded;

    private ServiceConnection conn = new ServiceConnection(){
        public void onServiceConnected(ComponentName cls, IBinder bnd){
            service = new Messenger(bnd);
            bounded = true;
        }

        public void onServiceDisconnected(ComponentName cls){
            service = null;
        }
    }
}
```

Bound Service con Messenger – Esempio IV

```
        bounded = false;
    }
};

@Override
protected void onCreate(Bundle savedInstanceState){
    /* ... */
}

@Override
protected void onStart(){
    super.onStart();

    Intent i = new Intent(this, MessengerService.class)
    bindService(i, conn, Context.BIND_AUTO_CREATE);
}
```

Bound Service con Messenger – Esempio V

```
@Override
protected void onStop(){
    super.onStop();





    if(bounded){
        unbindService(conn);
        bounded = false;
    }
}

public void requireServiceMessage1(View v){
    if (!bounded) return;

    Message msg = Message.obtain(null, MessengerService.
        MESSAGE_1_REQUEST, 0, 0);

    try{
        service.send(msg);
    } catch (RemoteException e){ /* ... */ }
}
```

Riferimenti - Risorse Online

-  **Android Developers - Guide**
» <https://developer.android.com/guide/>
-  **Android Developers - API Reference**
» <https://developer.android.com/reference/>
-  **Android Developers - Samples**
» <https://developer.android.com/samples/>
-  **Android Developers - Design & Quality**
» <https://developer.android.com/design/>



Riferimenti - Libri

-  Zigurd Mednieks, Laird Dornin, G. Blake Meike, Masumi Nakamura
Programming Android
O'Reilly, 2011
-  Chris Haseman, Kevin Grant
Beginning Android Programming: Develop and Design
Peachpit Press, 2013
-  Ronan Schwarz, Phil Dutson, James Steele, Nelson To
The Android Developer's Cookbook : Building Applications with the Android SDK
Addison-Wesley, 2013
-  Theresa Neil
Mobile Design Pattern Gallery: UI Patterns for Smartphone App
O'Reilly, Second Edition, 2014