Intents

Reminder Activities

Fino ad ora le nozioni trattate promuovono lo sviluppo di un'applicazione *mono-activity*, uno dei paradigmi più utilizzati all'interno di *Android*, dato soprattutto dall'abuso del metodo *SetContentView()*. Tuttavia, in certe circostanze, potrebbe essere necessario implementare più di una singola attività. Tutto questo si traduce in un ulteriore layer di difficoltà, in quanto le conoscenze possedute non permettono di manipolare un insieme di attività.

Proseguendo, la possibilità di navigare e, pertanto, di gestire molteplici *activity* è data dalla nozione **intent**. Innanzitutto occorre che l'insieme di *activities* sia dichiarato all'interno del *Manifest*, passo fondamentale affinchè sia possibile correlare le attività, per poi sviluppare gli *intenti*; tuttavia è bene prima soffermarsi sulle peculiarità di questo strumento.

Overview intent

Un *intent* rappresenta simbolicamente un *collante* tra attività durante *runtime*. Effettivamente è definito come un *oggetto messaggio*, poichè demanda ad Android di compiere determinate azioni rispetto a specifici dati inviati in input. Per cui, dalla breve definizione precedente, un *intent* è in grado di passare dati, chiamare componenti e di utilizzare funzionalità di applicazioni già installate.

Un *intento* è rappresentato graficamente come un bundle contenente informazioni, relative sia al *ricevitore* della comunicazione che al sistema *Android*.

```
val intent:Intent = Intent();
```

Inizializzazione di un intent

Esistono due tipi di *intent*, suddivisi in:

- **Intent explicit**, lo sviluppatore è a conoscenza del *receiver target* del componente demandato
- **Intent implicit**, lo sviluppatore non è a conoscenza del *receiver target* del componente demandato

Per cui, tramite i parametri inviati, il *sistema operativo* riesce a scindere sulla tipologia di *intent*; tutto dipende se sia dichiarato o meno il *Component Name*. Rispetto a quanto detto, susseguono due atteggiamenti distinti da parte del *operating system*, in cui:

- Nel primo caso il sistema operativo, interrogando il *Manifest*, è in grado di risalire allo specifico componente, risvegliandolo
- Contrariamente, nel secondo caso, l'*operating system* controlla tutte le attività dell'applicazioni installate, deducendo quali fra esse risponda alla richiesta formulata

Intent Description: Explicit

Intenti espliciti possiedono il *Component Name*; si tratta di un tag opzionale, ma all'interno di un ambiente simile la sua dichiarazione diviene obbligatoria.

```
intent.setComponent(ComponentName(
    this, // riferimento al contesto attuale
    MyActivity::class.java)
)
```

Inizializzazione di un intent

Affinchè sia possibile risvegliare una nuova attività, posta all'interno della stessa applicazione, si adotta la sintassi sottostante.

```
val intent:Intent = Intent(this, ActivityTwo::class.java)
startActivity(intent)
```

Start di una nuova activity

Tramite il metodo *startActivity()* è direttamente imposto che il componente gestito e manipolato dall'*intent* debba essere un'attività.

Intent Description: Implicit

Come già descritto precedentemente, *intenti impliciti* non sono caratterizzati da un *Component Name*. Questo testimonia un approccio differente rispetto alla propria controparte, in cui Android definisce quali componenti siano associabili all'*intent* specifico. In caso due o più *component* dovessero rispettare le aspettative richieste, sarà compito dell'user decidere quali fra essi debba essere utilizzato.

Field dell'Intent

Graficamente sono definiti sei field che contraddistinguono un *intent*, indifferentemente dalla tipologia. Di seguito è proposta una breve descrizione di ognuno di essi, omettendo il tag *Component Name*, dato che più volte è stata ribadita la sua nozione.

Action Name, si tratta semplicemente di una stringa nominativa dell'azione in questione. È obbligatoria nel caso di intenti impliciti, definita dallo sviluppatore oppure scelta tra le molteplici già disponibili.

```
intent.action = Intent.ACTION_EDIT // action name predefinito
intent.action = "com.example.MyApplication.MY_ACTION" // action name
    personalizzato
```

Definizione del field action name

L'impiego del field è attuato qualora si voglia imporre il comportamento che l'*activity* risvegliata debba eseguire

Data, rappresenta i dati passati dal chiamante al ricevitore. La sintassi utilizzata promuove una schema simile a quanto riportato.

```
intent.data = "https://www.unibo.it/"
intent.type = "text/html"
```

Definizone del field data

Prima di proseguire è bene evidenziare alcuni aspetti salienti dei due metodi. Innanzitutto le due funzioni si annullano a vicenda, pertanto qualora si abbia l'opportunità di modificare sia il *name* che il *type*, è necessario usufruire del metodo **setDataAndType()**. Il *name* coincide con l'*Uniform Resouce Identifier*, mentre il *type* testimonia il *Multipurpose Internet Mail Extensions*

Category, una stringa che aggiunge delle informazioni all'*action* da eseguire. Tipicamente è riportata in relazione ad *intent* che abbiano ulteriori *features* da considerare nella loro operatività.

```
intent.addCategory(Intent.CATEGORY_BROWSABLE)
```

Definizone del field category

Extra, informazioni aggiuntive precisate dal mandante, predisposte secondo la coppia chiave-valore.

```
val intent:Intent = Intent(Intent.ACTION_SEND)
intent.putExtra(Intent.EXTRA_MAIL, "federico.montori2@unibo.it")
```

Definizone del field extra

Extras possono essere predefiniti, ciò avviene per la maggiore delle *actions*, le quali richiedono che siano indicate tali informazioni aggiuntive

Flags, interi contenenti, nuovamente, informazioni aggiuntive che istruiscono *Android* in relazione all'approccio che debba mantenere nei confronti dei componenti risvegliati.

```
intent.flags = Intent.FLAG_ACTIVITY_NEW_TASK or
               Intent.FLAG_ACTIVITY_NO_ANIMATION
```

Definizione del field flags

Intent Resolution

Fino ad ora è stata piu volte ribadita la differenza principale tra un intento *esplicito* ed *implicito*; ciò dipende dal contenuto del bundle che li caratterizza, in cui, tipicamente, il primo citato è adottato qualora il *componente* risvegliato appartenga alla stessa applicazione, mentre nel secondo caso si attua qualora siano necessarie funzionalità provenienti da attività esterne all'applicazione sviluppata.

Tuttavia sorgono alcuni problemi, relativi a due contesti:

- Come riconoscere quale componente risponda alla richiesta, dato che Android rileva l'attività di destinazione in completa autonomia

- Identificare quali componenti possano gestire gli intenti creati

Rispetto alla prima problematica è necessario stabilire un esempio di supporto, definito come segue.

```
val intent:Intent = Intent(Intent.ACTION_SEND)
intent.putExtra(Intent.EXTRA_TEXT, "Hello World!")
intent.type = "text/plain"
if(intent.resolveActivity(packageManager) != null)
    startActivity(intent)
```

Esempio di creazione di un intent implicito

Si evidenziano alcuni aspetti descritti precedentemente, tra cui *Action Name*, *Extras* ed infine *Data*. Lo sketch di codice rappresenta appieno il quesito posto, in cui non è possibile rilevare quale componente comunichi dati. In tal senso è spesso attuato un paradigma, in cui l'utente è forzato a scegliere il componente che sarà successivamente risvegliato; a livello di codice quest'ultimo passaggio si traduce nell'inizializzazione di una variabile **chooser**.

```
val intent:Intent = Intent(Intent.ACTION_SEND)
intent.putExtra(Intent.EXTRA_TEXT, "Hello World!")
intent.type = "text/plain"
val chooser = Intent.createChooser(intent, "You have to choose!")
if(intent.resolveActivity(packageManager) != null)
    startActivity(chooser)
```

Inserimento dell'elemento chooser

Proseguendo, la seconda problematica è inerente al ricevitore della richiesta. Occorre quindi definire quali intenti il componente di riferimento possa gestire; ciò avviene attraverso l'utilizzo del tag *<intent-filter>/</intent-filter>* posto all'interno del *Manifest*.

```
<activity
    android:name = "MyApplication"
    android:exported = "true"
    <intent-filter>
        <action android:name = "com.example.ACTION_ECHO" />
    </intent-filter>
</activity>
```

Inserimento dell'intent-filter nel Manifest

Piccola peculiarità, è dettata dalla presenza dell'*attribute exported*; esso testimonia che l'*activity* può essere invocata da ulteriori applicazioni. Proprio per tale ragione è necessario esplicitare quali *intents* il componente sia in grado di manipolare.

Intent with result

Individuati i campi principali che carattetterizzano un *intent* e stabilito il paradigma a cui affidarsi, affinchè sia possibile gestire uno strumento di tale complessità, occorre definire come restituire dei risultati. Gli intenti possono essere formulati affinchè possano ricevere messaggi in risposta.

Principalmente, le attività sono invocate per ottenere un risultato; tuttavia è bene distinguere quale sia l'entità in attesa di ricevere un riscontro, rispetto a colui che abbia il compito di formulare una risposta alla richiesta ricevuta. Per cui sono contraddistinti due attori, ossia:

- **Sender side**, invoca la funzione *startActivityForResult()*

```
val ACTIVITY_CODE = 0
val intent: Intent = Intent(this, ActivityPrefiz::class.java)
startActivityForResult(intent, ACTIVITY_CODE)
...
override fun onActivityResult
    (requestCode: Int, resultCode: Int, data: Intent?) {

}
```

Sender acquisisce risultati da attività invocate

Di seguito sono descritti alcuni step su cui occorre soffermarsi. Innanzitutto il *sender* attua il metodo *startActivityForResult()*, dove è dato in input lo stesso *intent* dichiarato e inizializzato precedentemente; quest'ultimo passaggio è necessario per garantire al ricevitore la possibilità di sfruttare il *"collante"*, pur di fornire le informazioni richieste. Proseguendo, qualora l'*activity* completi le proprie operazioni, il mittente invoca la funzione *onActivityResult()*, acquisendo, nello stesso ordine proposto dal metodo, un *requestcode*, attuato per discernere sulla tipologia di risultato restituito, un *resultcode*, codice in grado di stabilire la correttezza o meno della risposta ricevuta, ed infine i *data*, ossia i campi extra dell'intento forniti dal destinatario

- **Receiver side**, invoca il metodo *setResult()*

```
val intent = getIntent()
setResult(RESULT_OK, intent)
intent.putExtra("response", "whatever yuo wanted")
finish()
```

Receiver fornisce dati rispetto alla richiesta ricevuta

Da canto suo, il *receiver* acquisisce l'*intent* che abbia risvegliato l'*activity* e va ad agire sul *bundle* di riferimento, modificando i campi che lo contraddistinguono. Il risultato è restituito solamente se invocato il metodo *finish()*

Con l'avvento di **AndroidX**, i metodi *startActivityForResult()* e *onActivityResult()* sono stati accantonati per attuare e adeguare le **Activity Result API**. Le *Activity Result API* forniscono i componenti per la registrazione, l'avvio e la gestione di un risultato una volta inviato dal sistema. Uno strumento del genere vanifica circostanze in cui il processo, conseguente ad un'*attività*, sia eliminato a causa di situazioni di insufficienza di memoria.

Per questo motivo, le *Activity Result* registrano le *callback* qualunque cosa accada, in maniera tale che siano usufruibili indipendentemente dal fatto che le attività e i processi siano ricreati oppure distrutti.

In tal senso, è fornita un'API per la registrazione delle *callback*; acquisendo un *ActivityResultContract* e un *ActivityResultCallback* è restituito un *Launcher*, utilizzato per avviare l'attività circoscritta.

A livello esemplificativo è possibile delineare

```
val mLauncher: ActivityResultLauncher<String> = registerForActivityResult(
        ActivityResultContracts.GetContent(),
        ActivityResultCallback<Uri?>(){uri: Uri? -> /* handle the uri */}
    )
```

Istanza di un launcher per le activities

GetContent(), è un costruttore di default di contratti attuato per ottenere il contenuto di un risultato; in questo caso si tratterà di un *Uri*, proveniente dall'attività richiamata

registerForActivityResult(), restituisce un *launcher* che potrà essere conseguentemente utilizzato

Android Permission System

Molto spesso intenti *impliciti* demandano funzionalità che potrebbero violare la privacy dell'utente. In caso l'applicazione sviluppata richieda dati sensibili oppure azioni restrittive, occorre che sia formulata una richiesta relativa ai **permessi**.

I *permessi* sono tipicamente dichiarati all'interno del *Manifest*, richiesti, nella maggioranza dei casi, durante il *run-time* dell'applicazione.

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Definito il *permesso* all'interno del *Manifest*, la fase successiva comprende la richiesta dello stesso qualora sia runnata l'applicazione; ciò può avvenire tramite le *Activity Result API*, attraverso l'utilizzo di un *launcher*, che possa avviare l'attività, tendenzialmente visualizzata a video come un pop-up centrato, ed acquisire il risultato dettato dall'utente.

```
val requestPermissionLauncher = registerForActivityResult(
    ActivityResultContracts.RequestPermission()
    ) {isGranted: Boolean ->
        if(isGranted) {
            // Permission is granted
        } else {
            // Permission is denied
        }
    }
```

WebView

Una *View* che mostri a schermo una *web page* necessita di uno specifico permesso, denominato *android.permission.INTERNET*, uno dei pochi che non richieda il riconoscimento durante *runtime*. Riconosciuto, l'applicativo diviene un browser di piccole dimensioni, poichè permette la navigazione solo nel dominio dettato.

Android garantisce il funzionamento descritto mediante l'utilizzo delle **WebView**; è importante non confondere la realtà descritta rispetto alle tipiche funzionalità condotte tramite il *browser*, in questa casistica le *web pages* sono attuate solamente per la visualizzazione del contenuto delle stesse, senza alcuna azione secondaria di navigazione.

Tipicamente, all'interno di una qualsiasi applicazione sviluppata, avviene l'*override* del *behavior* di ogni link all'interno della *WebView* tramite un *WebViewClient*, sviluppato come segue.

```
webView.webViewClient = object: WebViewClient() {
    override fun shouldOverrideUrlLoading(view: WebView?,
    request: WebResourceRequest?): Boolean {
        if (request?.url?.host == Uri.parse(WEBSITE).host) {
            // This is my website, let the webView handle it
            return false
        } else return super.shouldOverrideUrlLoading(view, request)
    }
}
```