Apuntes

Descargar estos apuntes

B3 Resumen Intents

Índice

Intents

Comunicación de Activitys a través de Intents

Intents. Son los que permiten la comunicación con el resto de componentes, también se pueden definir como la voluntad de realizar una acción o una tarea determinada, estas tareas pueden ser una llamada de teléfono o visualizar una página web entre otras.

Intent explícito

En el inicio de una nueva actividad se pueden dar varias circunstancias:

- Inicio de una actividad sin argumentos ni devolución de resultados:

```
var intento=Intent(applicationContext, Activity2::class.java)
startActivity(intento)
```

- Inicio de una actividad con argumentos y sin devolución de resultados. Podemos pasar información a las actividades añadiendo datos al objeto intent con el método putExtra:

```
var intento= Intent(applicationContext, Activity2::class.java)
intento.putExtra("DATO", "Este es el valor que se manda desde Main Activity")
startActivity(intento)
```

  En la clase invocada obtendremos los datos recibidos de la siguiente manera:

```
var intento=intent
findViewById<TextView>(R.id.textoActivity2).setText(intento.getStringExtra("DATO"))
```

Intent con devolución de resultado

Contratos predeterminados

```
class MainActivity:AppCompatActivity(){
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        //creo en contrato al crear la activity
        val contrato =
            registerForActivityResult(ActivityResultContracts.StartActivityForResult())
            {
                result -> if (result.resultCode == Activity.RESULT_OK) {
                    //recupero dato devuelto, en este ejemplo una cadena llamada "DATORETURN"
                    val cadena = result.data?.getStringExtra("DATO_RETORNO")
                }
            }

        findViewById<TextView>(R.id.textoActivity1).setOnClickListener(
            //Al darle al botón, lanzo el contrato
            intent = Intent(applicationContext, Activity2::class.java)
            //intent.putExtra(...) Si se quiera mandar algo
            contrato.launch(intent)
    }
}
```

> 📌 Se debe registrar la actividad para posibles resultados en el método *onCreate()*.Si el resultado es OK realizamos el código necesario, en este caso se ha extraído la información del intent recibido **`result.data`** .

En la clase **invocada** devolveremos los datos de la siguiente manera:

```
       textView.setOnClickListener {
                //creamos un intent
                val intentoResultado=Intent()
4               //añadimos el resultado,
5               //se debe llamar igual que el que espera recibir el contrato
                intentoResultado.putExtra("DATO_RETORNO","Este es el dato")
                //setResultado con el código y el intentResultado.
                // Esto lo recogerá el contrato creado antes
                setResult(Activity.RESULT_OK,intentoResultado)
        }
```

Intent Implícito

Básicamente los pasos para lanzar una actividad con un **intent implícito** serían los siguientes:

- Declaramos el intent con la acción apropiada (ACTION_VIEW, ACTION_WEB_SEARCH, etc).
- Adjuntamos información adicional necesaria dependiendo de la acción del intent.
- Lanzamos el intent dejando que el sistema encuentre la actividad adecuada.

Para iniciar la actividad utilizaremos igualmente startActivity. Algunos ejemplos de Intents Comunes:

- Para mostrar una web dentro de un navegador:

```
val intento = Intent(Intent.ACTION_VIEW,
        Uri.parse("http://www.google.es"));
startActivity(intento);
```

  Requiere los permisos: `<uses-permission android:name=" android.permission.INTERNET "/>`

- Para utilizar la búsqueda de google;

```
 val intento = Intent(Intent.ACTION_WEB_SEARCH);
 intento.putExtra(SearchManager.QUERY,
        "I.E.S. Doctor Balmis");
 startActivity(intento);
```

  Requiere los permisos `<uses-permission android:name="android.permission.INTERNET"/>`

- Para mostrar los contactos.

```
    val intento = Intent(Intent.ACTION_VIEW,
            Uri.parse("content://contacts/people/"));
startActivity(intento);
```

  Requiere los permisos: `<uses-permission android:name="android.permission.READ_CONTACTS"/>`

- Para abrir una aplicación externa

```
val intent= Intent();
intent.setComponent(
    ComponentName("com.ejemplos.myapplication",
    "com.ejemplos.myapplication.MainActivity"))
if (intent.resolveActivity(packageManager) != null)
    startActivity(intent)
```

- Tomar una foto con la Camara: el contraro debe ser creado en onCreate

```
fun creaContratoFoto () {
    resultadoCamara =
        registerForActivityResult(ActivityResultContracts.StartActivityForResult()) { res
                if (result.resultCode== RESULT_OK)
                    binding.imagen.background=
                    BitmapDrawable(resources,
                    (result.data?.extras?.get("data") as Bitmap))
            }
    }

fun tomarFoto(){
    if (ContextCompat.checkSelfPermission
            (applicationContext,
            Manifest.permission.CAMERA)
            != PackageManager.PERMISSION_GRANTED)
    {
        ActivityCompat.requestPermissions(this,
            arrayOf(Manifest.permission.CAMERA),
            RESPUESTA_CAMARA)
    }
    else lanzarIntentFoto()
}

fun lanzarIntentFoto()
{
    uriImagen = Uri.parse("imagen")
    val cameraIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
    resultadoCamara.launch(cameraIntent)
}
```

- Seleccionar una foto de la galeria

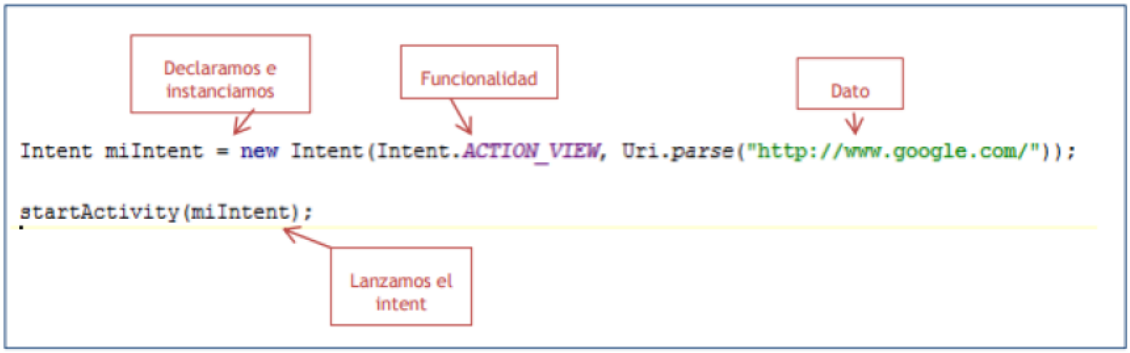
```
  resultadoGaleria =
        registerForActivityResult(
            ActivityResultContracts.StartActivityForResult())
            { result ->
                if (result.resultCode == Activity.RESULT_OK) {
                binding.imagen.setImageURI(result.data?.data)
                }
            }
```

- LLamada

```
 fun lanzarLlamada()
{
    if (ContextCompat.checkSelfPermission
            (applicationContext,
            Manifest.permission.CALL_PHONE)
        != PackageManager.PERMISSION_GRANTED)
        {
            ActivityCompat.requestPermissions(
                this,
                arrayOf(Manifest.permission.CALL_PHONE),
                RESPUESTA_LLAMADA)
        }
    else
    startActivity(Intent(Intent.ACTION_CALL,
        Uri.parse("tel:" + binding.tlfText.text.toString())))
}
```

- Lanzar correo

```
fun lanzarCorreo(email: String)
{
    val intent = Intent(Intent.ACTION_SEND)
    intent.type = "message/rfc822"
    intent.putExtra(Intent.EXTRA_EMAIL, arrayOf(email))
    startActivity(Intent.createChooser(intent, "Tipo mensaje"))
}
```



Es posible que en ocasiones no se pueda acceder a una aplicación debido a que no está instalada en el dispositivo, a restricciones de perfil o a la configuración de un administrador. Para verificar que una actividad recibirá la intent, es muy importante usar antes `resolveActivity()` en el objeto Intent. Si el resultado no es nulo, hay al menos una aplicación que puede administrar la intent y es seguro llamar a `startActivity()`.

```
val intento = Intent(Intent.ACTION_VIEW,
      Uri.parse("content://contacts/people/"));

if (intento.resolveActivity(packageManager) != null)
      startActivity(intento);
```

Requerir permisos al usuario

Solicitud de permisos

1. En el archivo de manifiesto de la app, declara los permisos que necesites.
2. Esperar a que el usuario invoque la tarea o acción de la app que requiere acceso a datos privados específicos. En ese momento, la app puede solicitar el permiso de tiempo de ejecución necesario para acceder a esos datos. Para ello:
   a. Verifica si el usuario ya otorgó el permiso de tiempo de ejecución que requiere la app. De ser así, esta ya puede acceder a los recursos necesarios.
   b. En caso contrario, solicitar el permiso en tiempo de ejecución, en ese caso el sistema mostrará un dialogo solicitando el permiso y si es otorgado se pasará al acceso de los recursos. En este paso se puede optar por mostrar un dialogo explicando porque motivo son necesarios los recursos.
3. Si el usuario rechaza la solicitud, la app debería proporcionar una opción elegante al usuario.

```
val RESPUESTA_PERMISOS = 111
@RequiresApi(Build.VERSION_CODES.Q)
fun solicitarPermisos()
{
    if (checkSelfPermission(READ_EXTERNAL_STORAGE) == PackageManager.PERMISSION_DENIED
        || checkSelfPermission(READ_CONTACTS) == PackageManager.PERMISSION_DENIED)
    {
        if (shouldShowRequestPermissionRationale(READ_EXTERNAL_STORAGE))
            //Si se decide explicar los motivos de los permisos con  similar a un dialogo
        if (shouldShowRequestPermissionRationale(READ_CONTACTS))
            //Si se decide explicar los motivos de los permisos con  similar a un dialogo
        //Con requestPermissions se pide al usuario que permita los permisos,
        //en este caso dos
        requestPermissions(arrayOf(READ_EXTERNAL_STORAGE, _CONTACTS), RESPUESTA_PERMISOS)
    }
    else lanzarFuncionalidadQueRequierePermisos()
}
```

```
val RESPUESTA_SOLICITUD_PERMISOS = 111
val RESPUESTA_SOLICITUD_PERMISOS_2 = 222
@RequiresApi(Build.VERSION_CODES.Q)
override fun onRequestPermissionsResult(
       requestCode: Int,//código de identificación del resultado
       permissions: Array<out String>,//array con los nombres de los permisos
       grantResults: IntArray//array de 0 y -1 (permitido, no permitido) en orden)
{
       super.onRequestPermissionsResult(requestCode, permissions, grantResults)
       when (requestCode) {
           RESPUESTA_SOLICITUD_PERMISOS -> {
               if (grantResults[0] == PackageManager.PERMISSION_GRANTED)
                   Toast.makeText(
                       applicationContext,
                       permissions[0] + " Permiso concedido",
                       Toast.LENGTH_SHORT
                   ).show()
               if (grantResults[1] == PackageManager.PERMISSION_GRANTED)
                  ...
            RESPUESTA_SOLICIUD_PERMISOS_2 ->
            .....
           }
       }
   }
```

> El hilo que controla la respuesta del usuario, se recoge mediante el método sobreescrito **`onRequestPermissionsResult`** , a este método le llega la información necesaria para poder saber a que petición de permisos se refiere (usando las constantes RESPUESTA_PERMISOS utilizadas en las peticiones), el array con los nombres de los permisos y el resultado de si han sido concedido o no.
> En cada petición de permisos puede haber un array con varios ermisos.
> permissions[0] será el primer permiso de esta solicitud, y grantResults[0] la respuesta a ese permiso. permissions[1] será el segundo permiso de esta solicitud, y grantResults[1] la respuesta a ese permiso.

Intent-filter

Cuando se crea un intent implícito el sistema Android busca el componente apropiado que pueda resolver la petición dada en el intent. La búsqueda la realiza analizando los **intent-filter** definidos en los archivos de manifiesto de las aplicaciones instaladas en el dispositivo.
A un Intent podemos asociarle una acción, unos datos y una categoría. Las actividades pueden declarar el tipo de acciones que pueden llevar a cabo y los tipos de datos que pueden gestionar.

- Las acciones son cadenas de texto estándar que describen lo que la actividad puede hacer. Por ejemplo **`android.intent.action.VIEW`**, es una acción que indica que la actividad puede

mostrar datos al usuario.

- La misma actividad puede declarar el tipo de datos del que se ocupa, por ejemplo `vnd.android.cursor.dir/person`, indica que la actividad manipula los datos de la agenda.
- También pueden declarar una categoría, que básicamente indica si la actividad va a ser lanzada desde el lanzador de aplicaciones, desde el menú de otra aplicación o directamente desde otra actividad.las actividades que están dispuestas a recibir intents implícitos deben incluir `android.intent.category.DEFAULT`.

Un posible ejemplo del AndroidManifest.xml que cumpla las anteriores condiciones:

```
    <intent-filter>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="vnd.android.cursor.dir/person"/>
    </intent-filter>
```

Una activity puede tener varios intent-filter definidos. Igualmente cada uno de esos intent-filter puede tener varias acciones definidas.

Declarar content provider para acceder a recursos(p.e.una canción)

1. Crear una carpeta xml en los recursos, `res\xml` y dentro un fichero, por ejemplo `provider_paths`. Donde añadiremos el siguiente código para referenciar al directorio raíz.

```
<?xml version="1.0" encoding="utf-8"?>
<paths>
    <external-path name="external_files" path="."/>
</paths>
```

2. Añadir en el Manifest un provider que haga referencia al recurso creado. Dentro de la etiqueta aplicación y teniendo en cuenta no olvidar los permisos de `READ_EXTERNAL_STORAGE`.

```
<provider
    android:name="androidx.core.content.FileProvider"
    android:authorities="${applicationId}.provider"
    android:exported="false"
    android:grantUriPermissions="true">
    <meta-data
        android:name="android.support.FILE_PROVIDER_PATHS"
        android:resource="@xml/provider_paths" />
</provider>
```

```
private fun escucharCancion() {
     //Ruta de la carpeta que usa android para guardar recursos
     //de la aplicación, consta del dominio y es privada a esta
     var directorio=getExternalFilesDir(null)?.absolutePath
     //Cogemos el inicio  de la ruta, que corresponde con
     //los archivos multimedia 'storage/emulated/0/'
     //Realizamos las modificaciones en directorio para adaptarlo a lo que necesitemos y c
     val data = File(directorio + "music/minions.mp3")
     val uri = FileProvider.getUriForFile(
         this@MainActivity,
         BuildConfig.APPLICATION_ID + ".provider",
         data
     )
     val intent = Intent(Intent.ACTION_VIEW)
     intent.setDataAndType(uri, "audio/.mp3")
     intent.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
     startActivity(intent)
 }
```