Desarrollo para dispositivos móviles con Android: almacenamiento.

Contenido

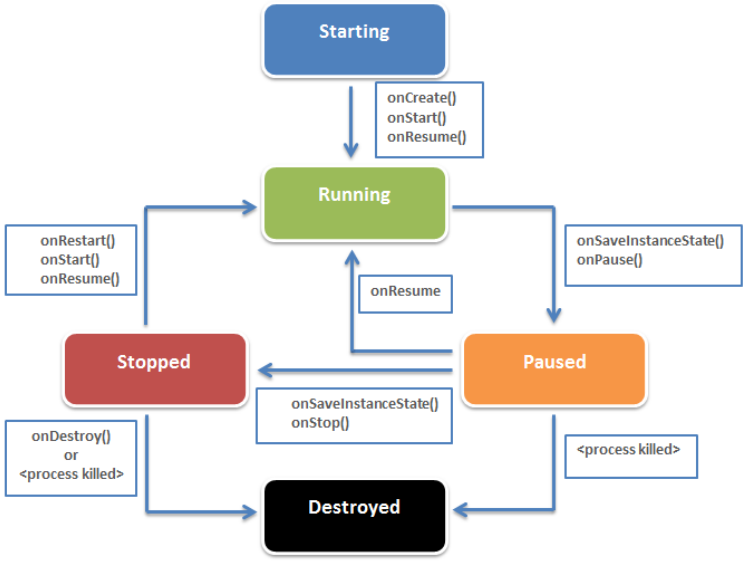
1 Introducción

En todas las aplicaciones es necesario salvaguardar datos, para que de alguna manera estas sean útiles. Android proporciona varios métodos para esto, a la vez que varias localizaciones distintas: almacenamiento interno y almacenamiento externo (tarjeta SD).

La forma más simple de almacenar valores es utilizar las posibilidades de almacenamiento de clave-valor que se conocen como preferencias. Además, es posible utilizar la posibilidad de crear archivos de configuración.

2 En qué momento guardar los datos

Las actividades pueden pausarse o pararse totalmente por diferentes factores, incluyendo: que se haya pulsado la tecla “atrás”, que se haya girado la pantalla o que se haya eliminado la aplicación en el administrador de tareas. Cuando esto sucede, ciertos métodos (que pueden se reescritos) son llamados.



Las aplicaciones se pausan cuando estas se ocultan parcialmente, por ejemplo debido a una notificación emergente o un acceso al menú superior. En este caso, la actividad (objeto **Activity**) en cuestión recibe una llamada al método *onPause()*, y al volver a obtener la atención del usuario, se llama a *onResume()*.

Las aplicaciones se paran por ejemplo, cuando se vuelve al menú principal, cuando se pulsa la tecla “atrás”, o cuando se gira la pantalla. Análogamente, se llama a los métodos *onStop()*, *onStart()* y *onRestart()* (este último nunca es llamado cuando la aplicación se ejecuta por primera vez).

En términos generales (como regla sencilla a recordar), es necesario guardar el estado en el método *onStop()*, y recuperarlo en *onStart()*. Sin embargo, y dado que por necesidades de recursos, Android puede eliminar aplicaciones sin garantía de que sus métodos *onStop()* sean llamados, es preferible hacerlo en los métodos *onPause()* y *onResume()*.

3 Utilizando las preferencias

La forma de comenzar a utilizar las preferencias varía según el método deseado. Depende de si se desea obtener un conjunto de referencias privado (lo más habitual) o público (menos habitual, en caso de que se quieran compartir las referencias entre varias aplicaciones).

```
SharedPreferences prefs = this.getPreferences( Context.MODE_PRIVATE );
```

La llamada anterior abre las preferencias privadas de la actividad actual, siendo accesible mediante *prefs*. Si solo se desea un almacenamiento de preferencias, entonces esta posibilidad es más que suficiente.

```
SharedPreferences prefs = this.getSharedPreferences( “mis_prefs”, Context.MODE_PRIVATE );
```

En el caso anterior, se abre un almacenamiento determinado por un nombre, lo cual es útil si se desean almacenar preferencias de manera compartimentada.

3.1 Leyendo preferencias

Una vez obtenida una referencia al almacenamiento deseado, es muy sencillo obtener un dato almacenado allí: existen distintos métodos para obtener distintos tipos de datos. Por ejemplo, para obtener un tipo de dato primitivo:

```
int x = prefs.getInt( “clave1”, valorPorDefecto );
float y = prefs.getFloat( “clave2”, valorPorDefecto );
String str = prefs.getString( “clave3”, valorPorDefecto );
```

Así, para recuperar por ejemplo un usuario y contraseña que se hayan guardado antes, se podría ejecutar lo siguiente:

```
String usr = prefs.getString( “loginUsr”, “” );
String pswd = prefs.getString( “loginPswd”, “” );
```

El tipo más complejo que admite **SharedPreferences** es **Set<String>**, un conjunto de cadenas. Por ejemplo, podría usarse para recuperar un conjunto de usuarios.

```
Set<String> usuarios = prefs.getStringSet( “loginUsrs”, new HashSet<String>() );
```

3.2 Escribiendo preferencias

La forma de escribir preferencias es, primero obtener un editor sobre ellas, y después aplicando un *commit()* (que guarda las preferencias de manera síncrona) o *apply()* (que hace lo mismo asíncronamente). Para escribir preferencias, se utilizan métodos *put()*, de manera análoga a la lectura[1].

```
SharedPreferences.Editor editor = prefs.edit();

editor.putString( “loginUsr”, usuario.getLogin() );
editor.putString( “loginPswd”, usuario.getPassword() );
editor.apply();
```

1 Es importante tener en cuenta que la información guardada en las preferencias no está encriptada.

4 Utilizando archivos

4.1 Almacenamiento interno

También es posible utilizar el almacenamiento interno o el almacenamiento externo. En este caso se verá cómo guardar archivos en el almacenamiento interno, que es privado a las aplicaciones e inaccesible excepto por la propia aplicación.

```
FileOutputStream fos = this.openFileOutput( “settings.xml”, Context.MODE_PRIVATE );
```

El archivo es creado, mediante la llamada anterior, en el almacenamiento interno. Es necesario recordar el nombre del archivo para poder accederlo en el futuro (si bien el método **Activity**.*fileList()* devolverá una lista de archivos creados por la aplicación).

Una vez obtenido el objeto **FileOutputStream**, es posible emplear todas las clases de la API de Java que se desee.

```
FileOutputStream file = this.openFileOutput( “settings.txt” );
PrintWriter printer = new PrintWriter( file );
printer.println( “Usuario: “ + usuario.getLogin() );
```

De manera análoga, se puede obtener un archivo para lectura.

```
FileInputStream file = this.openFileInput( “settings.txt” );
Usuario usuario;

try {
	BufferedReader reader = new BufferedReader( new InputStreamReader( file ) );
	String line = reader.getLine();

	usuario = new Usuario( line.substring( line.indexOf( ':' ) + 1 ).trim();
catch(FileNotFoundException exc) {
	Toast.makeText( this, “File not found”, Toast.LENGTH_LONG ).show();
}
```

Evidentemente, lo de arriba es tan solo un ejemplo, y nunca deberían leerse los archivos de preferencias de esta forma.

4.2 Almacenamiento externo

Una aplicación que requiera acceso al almacenamiento externo debe solicitar permiso para ello. Así, al comienzo del archivo *AndroidManifest.xml*, se debe incluir una línea como la que aparece a continuación:

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

La línea anterior implica permisos de lectura y escritura. En caso de necesitar solamente la lectura, se cambiaría la constante por **READ_EXTERNAL_STORAGE**.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.devbaltasarq.listacompra"
          android:versionCode="1"
          android:versionName="1.0">
    <uses-sdk android:minSdkVersion="21"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <application android:label="@string/app_name" android:icon="@drawable/ic_launcher">
        <activity android:name="com.devbaltasarq.listacompra.Main"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Pero solo este permiso no es suficiente. El almacenamiento externo puede tener habilitada solo la lectura, pero no la escritura. O puede no estar disponible (si es una tarjeta SD, puede haber sido retirada). Así que debe comprobarse el estado del almacenamiento con métodos como los siguientes:

```
/* Checks if external storage is available for read and write */
public boolean isExternalStorageWritable() {
    return Environment.MEDIA_MOUNTED.equals( Environment.getExternalStorageState() );
}

/* Checks if external storage is available to at least read */
public boolean isExternalStorageReadable() {
    String state = Environment.getExternalStorageState();

    return Environment.MEDIA_MOUNTED.equals( state )
        || Environment.MEDIA_MOUNTED_READ_ONLY.equals( state );
}
```

Hay varios directorios que son especiales para Android, como por ejemplo el de fotos, el de tonos de llamada, etc. Se puede acceder a estos directorios a través de las constantes presentes en **Environment**, y el método de la misma clase **Environment**.*getExternalStoragePublicDirectory()*. En la siguiente tabla se mencionan los más importantes. Nótese que los archivos en estos directorios no se eliminan cuando se desinstala la aplicación.

Constante	Significado
DIRECTORY_DCIM	Carpeta de fotos de la cámara
DIRECTORY_DOCUMENTS	Carpeta de documentos creados por el usuario.
DIRECTORY_DOWNLOADS	Carpeta de descargas.
DIRECTORY_MOVIES	Carpeta de películas del usuario.
DIRECTORY_PICTURES	Carpeta de imágenes del usuario.

Por ejemplo, de la siguiente forma se crea un archivo dentro de la carpeta de documentos, de manera que la aplicación puede escribir algo en él, y puede ser modificado posteriormente por el usuario o por cualquier otra aplicación.

```
private void writeSampleDoc(String contents) {
    // Access user's public documents directory file: hola.txt
    File file = new File(Environment.getExternalStoragePublicDirectory(
            Environment.DIRECTORY_DOCUMENTS), “hola.txt”);

    try ( FileOutputStream f = new FileOutputStream( file ) )
         {
        PrintStream out = new PrintStream( f );
        out.println( “hi!, hola, hello” );
        out.close();
    }
    catch(IOException exc) {
        Log.e( EtqApp, exc.getMessage() );
    }
}
```

Es posible, además, tener archivos de configuración en el almacenamiento externo que se eliminen cuando se desinstala la aplicación. Esto se logra a partir de una llamada a **Context**.*getExternalFilesDir(null)*. Nótese que por defecto estos archivos son accesibles y modificables por parte de cualquier aplicación.

```
private void writeSampleDoc(String contents) {
    // Access app's external “private” directory
    File file = new File( getExternalFilesDir(null), “hola.txt” );

    try ( FileOutputStream f = new FileOutputStream( file ) )
         {
        PrintStream out = new PrintStream( f );
        out.println( “hi!, hola, hello” );
        out.close();
    }
    catch(IOException exc) {
        Log.e( EtqApp, exc.getMessage() );
    }
}
```

5 Referencias

- Documentación y recursos de Android para desarrolladores (accedido en sept. 2015) http://developer.android.com/
- Ciclo de vida de una actividad: https://developer.android.com/training/basics/activity-lifecycle/
- Android: almacenamiento de datos https://developer.android.com/guide/topics/data/data-storage.html
- Ejemplo de App Android con preferencias: https://github.com/Baltasarq/DroidSeries/