

Tema 4

1. Persistencia
2. Servicios
3. Hilos
4. Internet
5. Ejercicios



Almacenamiento.

Tipos de Almacenamiento

- Preferencias
 - Almacén privado de datos simples en pares de clave y valor
- Almacenamiento Interno
 - Almacén de datos privados sobre la memoria del dispositivo
- Almacenamiento Externo
 - Almacén de datos público sobre el almacenamiento externo* compartido
- Base de datos. SQLite
 - Almacenar datos estructurados en una base de datos privada
- Internet
 - Almacenar datos en un servidor web



Almacenamiento.

*Almacenamiento Externo

Es el almacenamiento externo a las aplicaciones.

En dispositivos con SD el almacenamiento externo se hace en la SD.

En dispositivos sin SD es la partición (system/core/sdcard) que ocupa parte de la memoria interna.

Desde Android 6.0 es posible unificar la SD como memoria interna y su uso será tanto para almacenamiento interno como externo.

Diferenciamos entre almacenamiento externo (Portable /sdcard) y el externo usado como interno (Adopted, data/data/package_name).

Seguimos manteniendo el interno real (data/app/package_name)

<https://source.android.com/devices/storage/index.html>



Almacenamiento.

When external storage media is adopted, it's formatted and encrypted to only work with a single Android device at a time.

Because the media is strongly tied to the Android device that adopted it, it can safely store both apps and private data for all users.

Apps can be placed on adopted storage media only when the developer has indicated support through the `android:installLocation` attribute.

*New installs of **supported apps** are automatically placed on the storage device **with the most free space**, and users can move supported apps between storage devices in the Settings app.*

Apps moved to adopted media are remembered while the media is ejected, and return when the media is reinserted.



Almacenamiento.

Preferencias.

- Proporciona un mecanismo que permite guardar y recuperar los pares de clave y valor de forma persistente de los tipos de datos primitivos
- `getSharedPreferences()`
 - Almacena las preferencias en varios archivos identificados por su nombre, que se especifica con el primer parámetro
- `getPreferences()`
 - Almacena las preferencias en un archivo asociado a una «Activity».
 - Solo será accesible por dicha «Activity»



Almacenamiento.

Preferencias.

- Recuperar valores
 - SharedPreferences
 - getXXX()

```
// Restore preferences
SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
boolean silent = settings.getBoolean("silentMode", false);
setSilent(silent);
```



Almacenamiento.

Preferencias.

- Para editar los archivos de preferencias
 - SharedPreferences.Editor
 - edit()
 - putXXX()
 - commit()

```
// We need an Editor object to make preference changes.  
// All objects are from android.content.Context  
SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);  
SharedPreferences.Editor editor = settings.edit();  
editor.putBoolean("silentMode", mSilentMode);  
  
// Commit the edits!  
editor.commit();
```



Almacenamiento.

Almacenamiento Interno.

- Podemos guardar archivos directamente en el almacenamiento interno del dispositivo
- De forma predeterminada, los archivos guardados en la memoria interna son privados
- Otras aplicaciones no pueden acceder a ellos
- Cuando el usuario desinstala la aplicación, estos archivos se eliminan



Almacenamiento.

Almacenamiento Interno.

- Escritura
 - `openFileOutput()`
 - `FileOutputStream()`
 - `write()`
 - `close()`

```
String FILENAME = "hello_file";
String string = "hello world!";

FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_PRIVATE);
fos.write(string.getBytes());
fos.close();
```



Almacenamiento.

Almacenamiento Interno.

- `MODE_PRIVATE`
 - Archivo en modo de create/replace
 - Es el modo por defecto
 - El archivo creado sólo puede ser accedido por la aplicación
- `MODE_APPEND`
 - Tiene el ámbito del `MODE_PRIVATE`
 - Añade el contenido al final del fichero
- `MODE_WORLD_READABLE`
 - Archivo en modo create/replace
 - Puede acceder cualquier aplicación



Almacenamiento.

Almacenamiento Interno.

- Lectura
 - openFileInput()
 - FileInputStream()
 - read()
 - close()

```
String FICHERO = "documento";  
FileInputStream fis = openFileInput(FICHERO);  
byte[] buffer = new byte[1024];  
while(fis.read(buffer) > 0)  
    //Procesar fichero  
fis.close();
```



Almacenamiento.

Almacenamiento Interno.

- res/raw/
- openRawResources(R.raw.file)

```
InputStream is = getResources().openRawResource(R.raw.mi_fichero);  
byte[] buffer = new byte[1024];  
while(is.read(buffer) > 0)  
    //Procesar fichero  
is.close();
```

- getCacheDir()

```
File f = getCacheDir();  
FileOutputStream fos = new FileOutputStream(f);  
fos.write("Fichero temporal en cache".getBytes());
```



Almacenamiento.

Almacenamiento Interno.

- `getFilesDir()`
 - Obtiene la ruta completa del directorio de sistema de archivos donde se guardan los archivos internos
- `GetDir()`
 - Crea (o se abre una ya existente) una carpeta dentro del espacio interno de almacenamiento
- `DeleteFile()`
 - Elimina un archivo guardado en la memoria interna
- `fileList()`
 - Devuelve una matriz de los archivos guardados actualmente en la aplicación



Almacenamiento.

Almacenamiento Externo.

- Todos los dispositivos compatibles con Android disponen de un espacio compartido "de almacenamiento externo" donde guardar archivos
- Puede ser un medio de almacenamiento extraíble (como una tarjeta SD) o una memoria interna (no extraíble)
- Los archivos guardados en el almacenamiento externo son de lectura global y pueden ser modificados y eliminados por el usuario
- Los archivos externos pueden desaparecer si el usuario monta la unidad de almacenamiento externo en un equipo
- Todas las aplicaciones pueden leer y escribir ficheros ubicados en el de almacenamiento externo y el usuario puede eliminarlos



Almacenamiento.

Almacenamiento Externo.

- Verificar su disponibilidad

```
if (Environment.MEDIA_MOUNTED.equals(state)) {  
    // We can read and write the media  
    mExternalStorageAvailable = mExternalStorageWriteable = true;  
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {  
    // We can only read the media  
    mExternalStorageAvailable = true;  
    mExternalStorageWriteable = false;  
} else {  
    // Something else is wrong. It may be one of many other  
    // states, but all we need to know is we can neither read nor write  
    mExternalStorageAvailable = mExternalStorageWriteable = false;  
}
```



Almacenamiento.

Almacenamiento Externo.

- Acceder a ficheros de la aplicación
- `getExternalFilesDir()`
 - `NULL (/)`
 - `DIRECTORY_RINGTONES`
 - `DIRECTORY_MUSIC`
 - `DIRECTORY_PICTURES`
- Ficheros temporales
 - `getExternalCacheDir()`



Almacenamiento.

Base de Datos (SQLite).

- Android utiliza SQLite como base de datos
- Se recomienda el uso de la base de datos en hilos separados
- Es «Open Source»
- Es muy ligera «250k en memoria»
- Se usa en la mayoría de dispositivos embebidos
- No mantiene la integridad de los tipos
- No mantiene integridad referencial



Almacenamiento.

Base de Datos (SQLite).

Consideraciones

- No almacenar ficheros, almacenar su ruta
- Se recomienda usar un identificador autoincrementable en todas las tablas.

ContentValues

- Objeto utilizado para insertar datos en la base de datos
- Es un diccionario de pares clave valor
- La clave es el nombre de la columna
- El valor es el valor de la columna



Almacenamiento.

Base de Datos (SQLite).

Cursores

- Las peticiones en Android devuelven Cursores
- moveToFirst
 - Desplaza el cursor a la primera fila
- moveToLast
 - Desplaza el cursor a la última fila
- moveToPrevious
 - Desplaza el cursos a la fila anterior
- moveToNext
 - Desplaza el cursor a la siguiente fila
- getCount
 - Devuelve el numero de filas
- getXXX
 - Devuelve el valor de una columna



Almacenamiento.

Base de Datos (SQLite).

- Cursores
 - getColumnIndex()
 - Devuelve el índice de una columna

```
if(cursor != null && cursor.moveToFirst()){  
    do{  
        String nombre = cursor.getString(cursor.getColumnIndex("NOMBRE"));  
    }while(cursor.moveToNext());  
}
```



Almacenamiento.

Base de Datos (SQLite).

- Crear la Base de Datos
 - SQL

```
public interface Columns extends BaseColumns{  
    |  
    public static final String KEY_TITLE = "titulo_noticia";  
    public static final String KEY_BODY = "cuerpo_noticia";  
    public static final String KEY_LINK = "enlace_noticia";  
    public static final String KEY_POSTED = "fecha_noticia";  
  
    public static final int COLUMN_ID = 0;  
    public static final int COLUMN_TITLE = 1;  
    public static final int COLUMN_BODY = 2;  
    public static final int COLUMN_LINK = 3;  
    public static final int COLUMN_POSTED = 4;  
}
```



Almacenamiento.

Base de Datos (SQLite).

- Crear la Base de Datos
 - SQL

```
CREATE_TABLE_NEWS = "create table " + TABLE_NEWS + " ("
                    + _ID + " integer primary key autoincrement,"
                    + KEY_TITLE + " text not null,"
                    + KEY_BODY + " text not null,"
                    + KEY_LINK + " text not null,"
                    + KEY_POSTED + " long);";
```



Almacenamiento.

Base de Datos (SQLite).

- Crear la Base de Datos
 - SQLiteOpenHelper
 - Contexto para acceder a la base de datos
 - Nombre de la base de datos (fichero sqlite3)
 - CursorFactory
 - Modifica la clase cursor utilizada para obtener los recursos
 - Versión de la base de datos

```
private class NewsDBHelper extends SQLiteOpenHelper {  
  
    public NewsDBHelper(Context context, String name,  
        CursorFactory factory, int version) {  
        super(context, name, factory, version);  
        // TODO Auto-generated constructor stub  
  
    }  
}
```



Almacenamiento.

Base de Datos (SQLite).

- Crear la Base de Datos
 - SQLiteOpenHelper
 - onCreate()
 - Si la BD no existe la crea y nos devuelve una instancia
 - Ejecutamos el script de creación

```
@Override
public void onCreate(SQLiteDatabase db) {
    // TODO Auto-generated method stub
    db.execSQL(CREATE_TABLE_NEWS);
}
```



Almacenamiento.

Base de Datos (SQLite).

- Crear la Base de Datos
 - SQLiteOpenHelper
 - onUpdate()
 - Si la versión de la BD no coincide con la actual
 - Proporciona la versión existente y la versión solicitada

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    // TODO Auto-generated method stub
    db.execSQL(DROP_TABLE_NEWS);
    onCreate(db);
}
```



Almacenamiento.

Base de Datos (SQLite).

- Obtener la base de datos
 - Iniciar SQLiteOpenHelper

```
public NoticiasDBAdapter(Context ctx) {  
    // TODO Auto-generated constructor stub  
    mContext = ctx;  
    mNewsHelper = new NewsDBHelper(mContext, DB_NAME, null, DB_VERSION);  
}
```



Almacenamiento.

Base de Datos (SQLite).

- Obtener la base de datos
 - Abrir base de datos

```
try{  
    mNewsDB = mNewsHelper.getWritableDatabase();  
}catch (SQLException e) {  
    // TODO: handle exception  
    mNewsDB = mNewsHelper.getReadableDatabase();  
}
```

- Cerrar base de datos

```
public void closeDB() {  
    mNewsDB.close();  
}
```



Almacenamiento.

Base de Datos (SQLite).

- Realizar consultas
 - El nombre de la tabla a consultar
 - Un array con las columnas que queremos recuperar. Null para recuperar todas
 - Clausula «WHERE» con formato «campo = ?»
 - Array de valores para la clausula «where»
 - Clausula «GROUP BY»
 - Clausula «Having»
 - Clausula de ordenamiento del resultado
 - DESC
 - ASC
 - Clausula del limite de resultados



Almacenamiento.

Base de Datos (SQLite).

- Realizar consultas

```
Cursor c = mNewsDB.query(TABLE_NEWS, null, _ID + "=" + id, null, null, null, null);
```

```
String[] projection = {_ID, KEY_TITLE};
```

```
String order = KEY_POSTED + " DESC";
```

```
String limit = "LIMIT 0,20";
```

```
return mNewsDB.query(TABLE_NEWS, projection, null, null, null, order, limit);
```

```
String[] projection = {_ID, KEY_TITLE};
```

```
String order = KEY_POSTED + " DESC";
```

```
String limit = "LIMIT 0,20";
```

```
Date now = new Date();
```

```
long today = new Date(now.getYear(), now.getMonth(), now.getDay()).getTime();
```

```
return mNewsDB.query(TABLE_NEWS, projection, KEY_POSTED + " >= " + today, null, null, order, limit);
```



Almacenamiento.

Base de Datos (SQLite).

- Insertar registros
 - Insert()
 - Nombre de la tabla
 - Campo que queremos poner a null (opcional, si usamos un contentValues vacío)
 - Valores de los campos

```
public long insertNews(Noticia noticia){  
    ContentValues cv = noticia.getValues();  
    long id = mNewsDB.insert(TABLE_NEWS, null, cv);  
    noticia.setId(id);  
    return id;  
}
```



Almacenamiento.

Base de Datos (SQLite).

- Actualizar registros
 - Update()
 - Nombre de la tabla
 - Nuevos valores de los campos
 - Condición «where»
 - Valores de la condición «where»

```
public int updateNews(Noticia noticia){  
    String where = "_ID" + "=" + noticia.getId();  
    ContentValues cv = noticia.getValues();  
    return mNewsDB.update(TABLE_NEWS, cv, where, null);  
}
```



Almacenamiento.

Base de Datos (SQLite).

- Eliminar registros
 - delete()
 - Nombre de la tabla
 - Condición «where»
 - Valores de la condición «where»

```
public boolean removeNews(Noticia noticia){  
    String where = _ID + "=" + noticia.getId();  
    return mNewsDB.delete(TABLE_NEWS, where, null) > 0;  
}
```



Almacenamiento.

Base de Datos (SQLite).

- `execSQL()`
 - Ejecuta cualquier consulta excepto `SELECT`
 - No se permite multiples consultas separadas por «;»
 - No devuelve datos
- `rawQuery()`
 - Ejecuta una consulta SQL y devuelve un cursor
 - No tiene limitaciones a las consultas
 - La condición «WHERE» usa el formato «campo = ?»



Almacenamiento.

Sugar ORM

```
compile 'com.github.satyan:sugar:1.4'
```

```
<application android:label="@string/app_name" android:icon="@drawable/icon"
android:name="com.orm.SugarApp">
.
.
<meta-data android:name="DATABASE" android:value="sugar_example.db" />
<meta-data android:name="VERSION" android:value="2" />
<meta-data android:name="QUERY_LOG" android:value="true" />
<meta-data android:name="DOMAIN_PACKAGE_NAME" android:value="com.example" />
.
.
</application>
```



Almacenamiento.

Sugar ORM

```
public class Book extends SugarRecord {  
    String title;  
    String edition;  
  
    public Book(){  
    }  
  
    public Book(String title, String edition){  
        this.title = title;  
        this.edition = edition;  
    }  
}
```



Almacenamiento.

Sugar ORM

```
@Table
public class Book {
    private Long id;

    public Book(){
    }

    public Book(String title, String edition){
        this.title = title;
        this.edition = edition;
    }

    public Long getId() {
        return id;
    }
}
```



Almacenamiento.

Sugar ORM

```
@Table
public class Book {
    private Long id;

    public Book(){
    }

    public Book(String title, String edition){
        this.title = title;
        this.edition = edition;
    }

    public Long getId() {
        return id;
    }
}
```



Consideraciones Previas.

Hilo Principal

- UI thread
- Encargado de procesar los eventos
- Encargado de actualizar las vistas
- ANR Android Not Responding (~5s)

!! Evitar bloquear el UI Thread !!



Consideraciones Previas.

Hilo Principal

- Modelo Single-Threaded

```
public void onClick(View v) {  
    new Thread(new Runnable() {  
        public void run() {  
            Bitmap b = loadImageFromNetwork();  
            mImageView.setImageBitmap(b);  
        }  
    }).start();  
}
```



Consideraciones Previas.

Hilo Principal

- View.Post()

```
public void onClick(View v) {  
    new Thread(new Runnable() {  
        public void run() {  
            final Bitmap b = loadImageFromNetwork();  
            mImageView.post(new Runnable() {  
                public void run() {  
                    mImageView.setImageBitmap(b);  
                }  
            });  
        }  
    }).start();  
}
```



Consideraciones Previas.

Hilo Principal

- Activity.runOnUiThread()

```
new Thread(new Runnable() {  
  
    public void run() {  
        // TODO Auto-generated method stub  
  
        //Trabajo pesado  
  
        AplicacionActivity.this.runOnUiThread(new Runnable() {  
            public void run() {  
                // TODO Auto-generated method stub  
                //Volcado en el UI thread  
            }  
        });  
    }  
}).start();
```



Consideraciones Previas.

Hilo Principal

- View.postDelayed()

```
new Thread(new Runnable() {  
  
    public void run() {  
        // TODO Auto-generated method stub  
  
        //Trabajo pesado  
  
        mIvPhoto.postDelayed(new Runnable() {  
  
            public void run() {  
                // TODO Auto-generated method stub  
                //Volcado en el UI  
            }  
        }, 1500);  
    }  
}).start();
```



Consideraciones Previas.

Hilo Principal

- Handler

```
final Handler handler = new Handler(new Handler.Callback() {  
  
    public boolean handleMessage(Message msg) {  
        // TODO Auto-generated method stub  
        //Volrcar cambios en la UI  
        return false;  
    }  
});  
  
new Thread(new Runnable() {  
  
    public void run() {  
        // TODO Auto-generated method stub  
  
        //Trabajo pesado  
  
        handler.sendMessage(handler.obtainMessage());  
    }  
}).start();
```



Consideraciones Previas.

Hilo Principal

- AsyncTask<A,B,C>
 - execute()
 - onPreExecuted()
 - UI Thread
 - doInBackground()
 - Thread propio
 - publishProgress()
 - onProgressUpdate()
 - UI Thread
 - onPostExecute()
 - UI Thread



Consideraciones Previas.

Hilo Principal

- AsyncTask<A,B,C>

```
new Downloader().execute(new URL("http://..."));
```



```
class Downloader extends AsyncTask<URL, Integer, String>{  
    @Override  
    protected void onPreExecute() {  
        // TODO Auto-generated method stub  
        super.onPreExecute();  
    }  
  
    @Override  
    protected String doInBackground(URL... params) {  
        // TODO Auto-generated method stub  
        publishProgress(5);  
        return new String();  
    }  
  
    @Override  
    protected void onProgressUpdate(Integer... values) {  
        // TODO Auto-generated method stub  
        super.onProgressUpdate(values);  
    }  
  
    @Override  
    protected void onPostExecute(String result) {  
        // TODO Auto-generated method stub  
        super.onPostExecute(result);  
    }  
}
```



Servicios

- Permite realizar procesos “Pesados” en segundo plano
- No tiene interfaz gráfica
- Puede ser iniciado por cualquier componente
 - Activity
 - Broadcast
 - Otro Servicio
- Continúa en funcionamiento aunque finalice el componente que lo inició
- Permite mecanismos IPC para comunicarse con otros componentes



Servicios

- “Started”
 - `startService()`
 - Queda en background de forma indefinida
 - Solo se cierra al finalizar su trabajo
 - Debe cerrarse así mismo
- “Bound”
 - `bindService()` Usa mecanismos IPC
 - Solo funciona mientras tiene una aplicación “unida”
 - Puse ser unido a mas de una aplicación



Servicios

- Heredamos de la clase “Service”
- onStartCommand()
 - Se llama cuando una aplicación quiere iniciar el servicio
 - startService()
 - Es responsabilidad del programador detenerlo
 - stopService()
 - stopSelf()
- Onbind()
 - Se ejecuta cuando una aplicación quiere unirse a un servicio
 - bindService()
 - La comunicación se realiza mediante IBinder

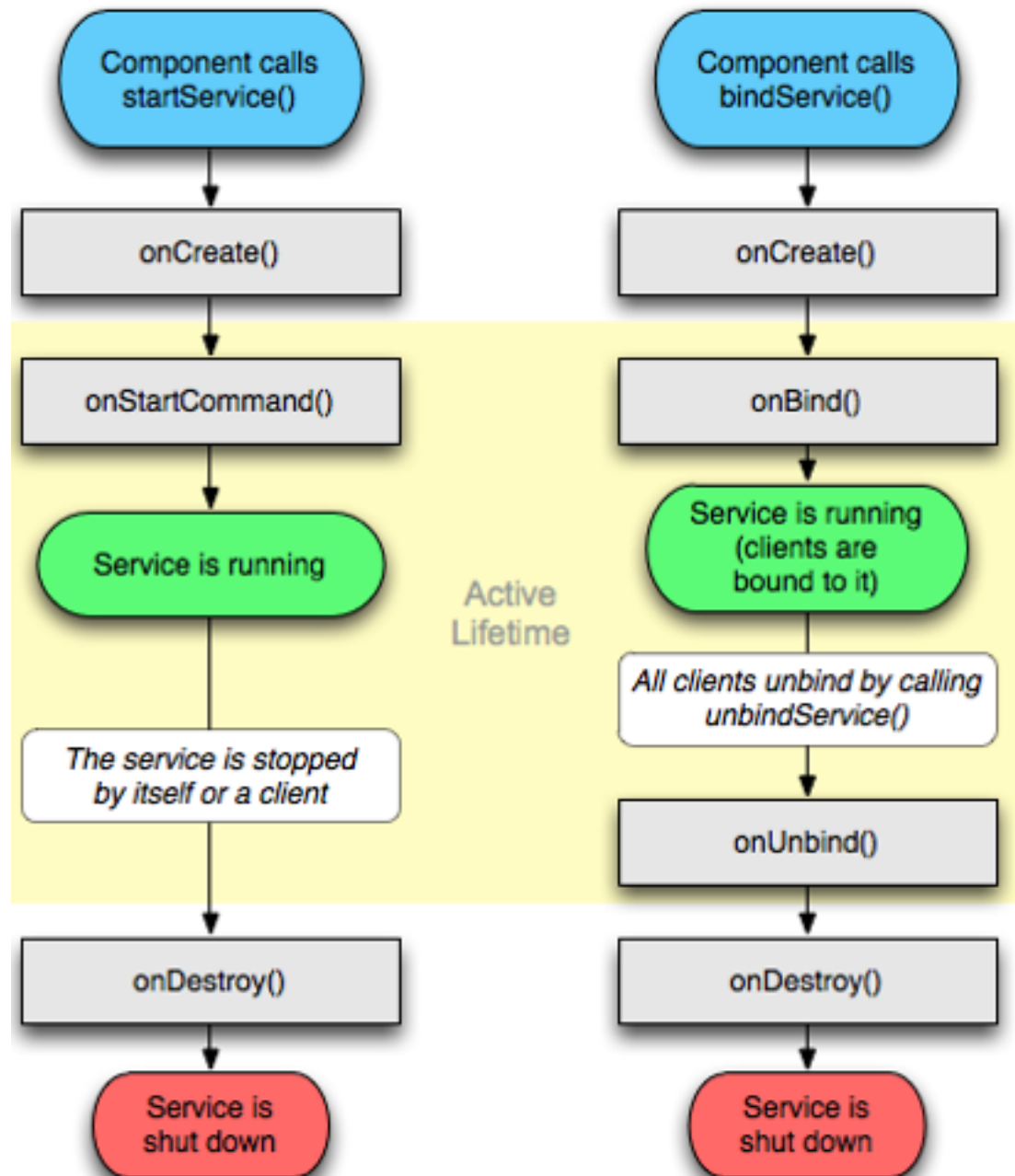


Servicios

- onCreate()
 - Se ejecuta la primera vez que creamos un servicio
 - Si el servicio ya está en marcha es llamado
- onDestroy()
 - Se ejecuta al destruir el servicio
- Android fuerza la parada de un servicio sólo si necesita memoria
- Los servicios “started” tienen preferencia para ser detenidos sobre los “bound” si alguna de las aplicaciones que tiene unidas esta en primer plano
- Debemos implementar mecanismos “elegantes” para relanzar servicios “started” finalizados por el sistema



Servicios



Servicios

- Declarar el servicio en el “Manifest”
 - `Android:exported=“true|false”`
 - Servicio publico|privado

```
<manifest ... >
    ...
    <application ... >
        <service android:name=".ExampleService" />
        ...
    </application>
</manifest>
```

```
<service android:exported="false" android:name=".services.SampleService"/>
```



Servicios “Started”

- Se lanza con `startService`
- Ejecuta `onStartCommand()`
- Debemos detenerlo manualmente
 - `stopService()`
 - `stopSelf()`
- Los procesos pesados del servicio debemos instanciarlos en un hilo nuevo



Servicios “Started”

- Heredamos de “IntentService”
 - Se basa en el paso de “Intents”
 - Crea su propio hilo de ejecución
 - Gestiona los temas de threading
 - Gestiona la finalización del servicio una vez que ha procesado todos los “Intents” recibidos
 - `onHandleIntent()`
 - Proceso en un hilo independiente que gestiona cada “Intent” recibido
- Solo necesitamos implementar el método `onHandleIntent()`



```
public class HelloIntentService extends IntentService {

    /**
     * A constructor is required, and must call the super IntentService(String)
     * constructor with a name for the worker thread.
     */
    public HelloIntentService() {
        super("HelloIntentService");
    }

    /**
     * The IntentService calls this method from the default worker thread with
     * the intent that started the service. When this method returns, IntentService
     * stops the service, as appropriate.
     */
    @Override
    protected void onHandleIntent(Intent intent) {
        // Normally we would do some work here, like download a file.
        // For our sample, we just sleep for 5 seconds.
        long endTime = System.currentTimeMillis() + 5*1000;
        while (System.currentTimeMillis() < endTime) {
            synchronized (this) {
                try {
                    wait(endTime - System.currentTimeMillis());
                } catch (Exception e) {
                }
            }
        }
    }
}
```

Servicios “Started”

- Heredar de “Service”
 - onCreate()
- onStartCommand()
 - START_STICKY
 - Vuelve a levantar el servicio mandando un “Intent” nulo a startCommand()
 - START_NO_STICKY
 - No vuelve a levantar el servicio
 - START_REDELIVERED_INTENT
 - Vuelve a levantar el servicio mandando el ultimo “Intent” recibido a startCommand()
- onDestroy()



Servicios “Started”

```
public class SampleService extends Service {  
  
    @Override  
    public void onCreate() {  
        // TODO Auto-generated method stub  
        super.onCreate();  
    }  
  
    @Override  
    public void onDestroy() {  
        // TODO Auto-generated method stub  
        super.onDestroy();  
    }  
}
```



```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    // TODO Auto-generated method stub
    //Gestion manual del threading
    new Thread(new Runnable() {

        public void run() {
            // TODO Auto-generated method stub
            //Trabajo pesado
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            lanzarNotificacion(getNotificacion());
            //detener servicio
            stopSelf();
        }
    }).start();

    return START_STICKY;
}
```



Servicios “Started”

- Lanzar el servicio

```
Intent intent = new Intent(this, HelloService.class);  
startService(intent);
```



Servicios “Bounded”

- Es un servicio que actúa como servidor
- Permite a otros componentes unirse a él
- Dispone de un sistema de comunicación con otros componentes IPC
- No se ejecuta de forma indefinida
- Suelen alimentar a otras aplicaciones
 - DownloadManager
 - UploadManager



Servicios “Bounded”

- Creamos la instancia de Binder
- Contiene los métodos que puede ejecutar el cliente
- Devuelve una instancia del servicio al que se une el componente



```
public class LocalService extends Service {
    // Binder given to clients
    private final IBinder mBinder = new LocalBinder();
    // Random number generator
    private final Random mGenerator = new Random();

    /**
     * Class used for the client Binder. Because we know
     * this class runs in the same process as its clients, we don't
     * need to do any special security checks.
     */
    public class LocalBinder extends Binder {
        LocalService getService() {
            // Return this instance of LocalService so the client
            // can call methods on it.
            return LocalService.this;
        }
    }

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

    /** method for clients */
    public int getRandomNumber() {
        return mGenerator.nextInt(100);
    }
}
```



Servicios “Bounded”

- Creamos la conexión entre el componente y el servicio

```
/** Defines callbacks for service binding, passed to bindService() */
private ServiceConnection mConnection = new ServiceConnection() {

    @Override
    public void onServiceConnected(ComponentName className,
        IBinder service) {
        // We've bound to LocalService, cast the IBinder and get Local
        LocalBinder binder = (LocalBinder) service;
        mService = binder.getService();
        mBound = true;
    }

    @Override
    public void onServiceDisconnected(ComponentName arg0) {
        mBound = false;
    }
};
```



Servicios “Bounded”

- Vincular y desvincular el servicio

```
@Override
protected void onStart() {
    super.onStart();
    // Bind to LocalService
    Intent intent = new Intent(this, LocalService.class);
    bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
}

@Override
protected void onStop() {
    super.onStop();
    // Unbind from the service
    if (mBound) {
        unbindService(mConnection);
        mBound = false;
    }
}
```



Servicios “Bounded”

- Obtener información del servicio

```
int num = mService.getRandomNumber();  
Toast.makeText(this, "number: " + num, Toast.LENGTH_SHORT).show();
```



Broadcast

- Es el sistema para propagar los mensajes del sistema operativo
- Registrarlos en el Manifest
 - Recibimos los mensajes aunque tengamos la aplicación cerrada
- Registrarlo en código
 - Registrar en el `onResume()`
 - Desregistrar en el `onPause()`
- No podemos mostrar dialogos
- No podemos unirnos a servicios
- Podemos iniciar un servicio
- Podemos levantar una aplicación
 - No se recomienda
 - Solo llamadas entrantes
- Podemos lanzar una notificación
 - Es lo más recomendado



Broadcast

- Registrar en el “Manifest”

```
<receiver android:name=".receiver.CustomReceiver">  
    <intent-filter>  
        <action android:name="com.juanjofp.android.CUSTOM_RECEIVER"/>  
    </intent-filter>  
</receiver>
```

```
public class CustomReceiver extends BroadcastReceiver{  
  
    public static final String CUSTOM_RECEIVER = "com.juanjofp.android.CUSTOM_RECEIVER";  
  
    @Override  
    public void onReceive(Context ctx, Intent data) {  
        // TODO Auto-generated method stub  
        Toast.makeText(ctx, "Recibido CustomReceiver", Toast.LENGTH_LONG).show();  
    }  
}
```

Broadcast

- Registrar desde código

```
public void registerReceiver(View v){
    mCustomReceiver2 = new CustomReceiver2();
    registerReceiver(mCustomReceiver2, new IntentFilter(CustomReceiver2.CUSTOM_RECEIVER2));
}

public void unregisterReceiver(View v){
    unregisterReceiver(mCustomReceiver2);
}

public class CustomReceiver2 extends BroadcastReceiver {

    public static final String CUSTOM_RECEIVER2 = "String com.juanjofp.android.CUSTOM_RECEIVER2";

    @Override
    public void onReceive(Context ctx, Intent data) {
        // TODO Auto-generated method stub
        Toast.makeText(ctx, "Recibido CustomReceiver2", Toast.LENGTH_LONG).show();
    }
}
```

Broadcast

- Lanzar el mensaje

```
public void sendCustomreceiver(View v){  
    sendBroadcast(new Intent (CustomReceiver.CUSTOM_RECEIVER));  
    sendBroadcast(new Intent (CustomReceiver2.CUSTOM_RECEIVER2));  
}
```



Broadcast del sistema

```
<receiver android:name=".services.SampleSystemReceiver">  
  <intent-filter>  
    <action android:name="android.intent.action.ACTION_POWER_CONNECTED"/>  
    <action android:name="android.intent.action.PHONE_STATE"/>  
    <action android:name="android.net.wifi.STATE_CHANGE"/>  
    <action android:name="android.intent.action.BATTERY_CHANGED"/>  
    <action android:name="android.intent.action.BOOT_COMPLETED"/>  
  </intent-filter>  
</receiver>
```



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

```
public void myClickHandler(View view) {  
    ...  
    ConnectivityManager connMgr = (ConnectivityManager)  
        getSystemService(Context.CONNECTIVITY_SERVICE);  
    NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();  
    if (networkInfo != null && networkInfo.isConnected()) {  
        // fetch data  
    } else {  
        // display error  
    }  
    ...  
}
```



```
// When user clicks button, calls AsyncTask.
// Before attempting to fetch the URL, makes sure that there is a network connection.
public void myClickHandler(View view) {
    // Gets the URL from the UI's text field.
    String urlString = urlText.getText().toString();
    ConnectivityManager connMgr = (ConnectivityManager)
        getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
    if (networkInfo != null && networkInfo.isConnected()) {
        new DownloadWebpageTask().execute(stringUrl);
    } else {
        textView.setText("No network connection available.");
    }
}

// Uses AsyncTask to create a task away from the main UI thread. This task takes a
// URL string and uses it to create an HttpURLConnection. Once the connection
// has been established, the AsyncTask downloads the contents of the webpage as
// an InputStream. Finally, the InputStream is converted into a string, which is
// displayed in the UI by the AsyncTask's onPostExecute method.
private class DownloadWebpageTask extends AsyncTask<String, Void, String> {
    @Override
    protected String doInBackground(String... urls) {

        // params comes from the execute() call: params[0] is the url.
        try {
            return downloadUrl(urls[0]);
        } catch (IOException e) {
            return "Unable to retrieve web page. URL may be invalid.";
        }
    }
    // onPostExecute displays the results of the AsyncTask.
    @Override
    protected void onPostExecute(String result) {
        textView.setText(result);
    }
}
```




```
// Given a URL, establishes an HttpURLConnection and retrieves
// the web page content as a InputStream, which it returns as
// a string.
private String downloadUrl(String myurl) throws IOException {
    InputStream is = null;
    // Only display the first 500 characters of the retrieved
    // web page content.
    int len = 500;

    try {
        URL url = new URL(myurl);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();
        conn.setReadTimeout(10000 /* milliseconds */);
        conn.setConnectTimeout(15000 /* milliseconds */);
        conn.setRequestMethod("GET");
        conn.setDoInput(true);
        // Starts the query
        conn.connect();
        int response = conn.getResponseCode();
        Log.d(DEBUG_TAG, "The response is: " + response);
        is = conn.getInputStream();

        // Convert the InputStream into a string
        String contentAsString = readIt(is, len);
        return contentAsString;

        // Makes sure that the InputStream is closed after the app is
        // finished using it.
    } finally {
        if (is != null) {
            is.close();
        }
    }
}
```



```
InputStream is = null;  
...  
Bitmap bitmap = BitmapFactory.decodeStream(is);  
ImageView imageView = (ImageView) findViewById(R.id.image_view);  
imageView.setImageBitmap(bitmap);
```

```
// Reads an InputStream and converts it to a String.  
public String readIt(InputStream stream, int len) throws IOException, UnsupportedEncodingException {  
    Reader reader = null;  
    reader = new InputStreamReader(stream, "UTF-8");  
    char[] buffer = new char[len];  
    reader.read(buffer);  
    return new String(buffer);  
}
```



Volley

```
dependencies {  
    ...  
    compile 'com.android.volley:volley:1.0.0'  
}
```

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



Volley

```
final TextView mTextView = (TextView) findViewById(R.id.text);
...

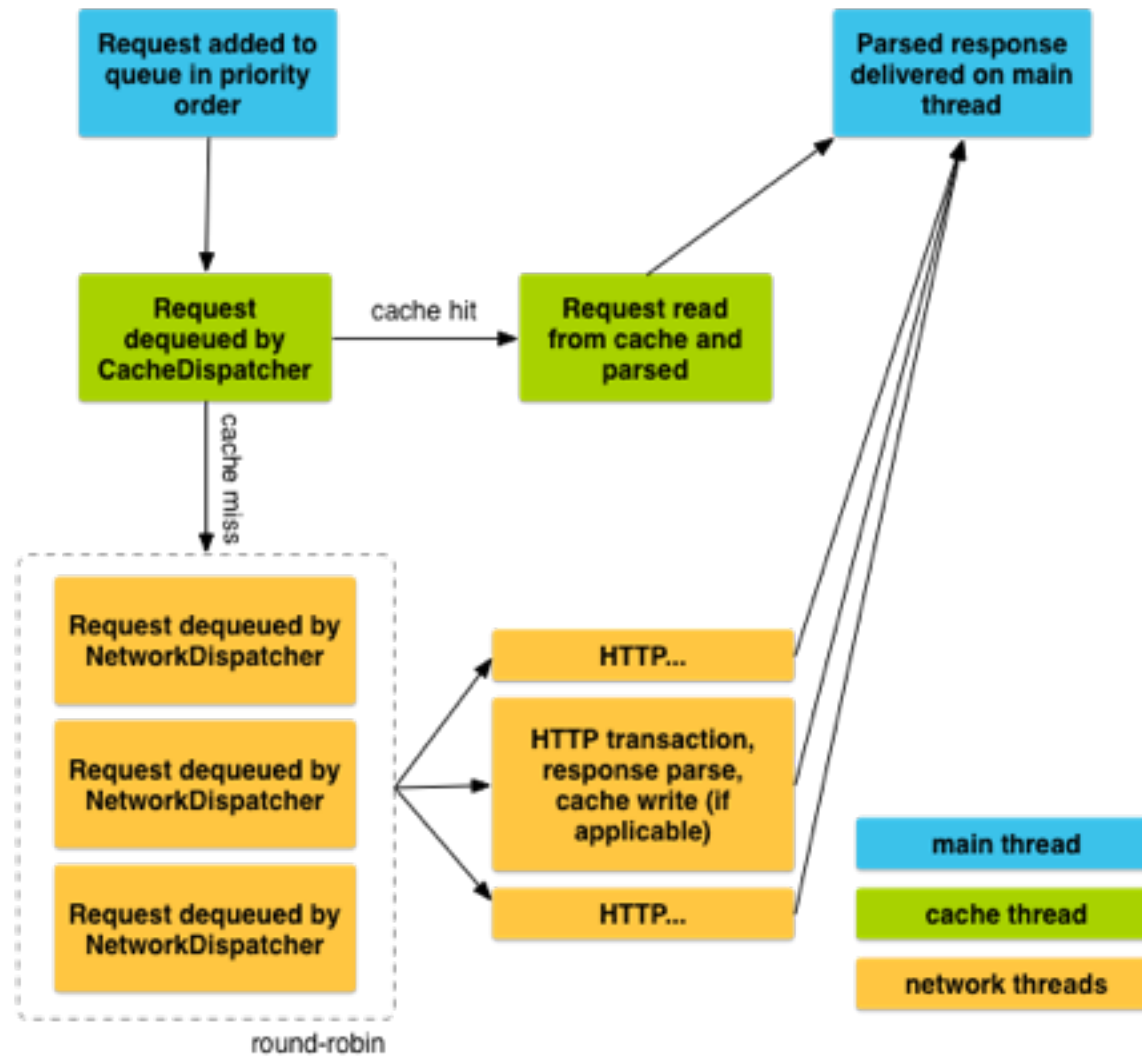
// Instantiate the RequestQueue.
RequestQueue queue = Volley.newRequestQueue(this);
String url = "http://www.google.com";

// Request a string response from the provided URL.
StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
    new Response.Listener<String>() {
        @Override
        public void onResponse(String response) {
            // Display the first 500 characters of the response string.
            mTextView.setText("Response is: " + response.substring(0,500));
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            mTextView.setText("That didn't work!");
        }
    });

// Add the request to the RequestQueue.
queue.add(stringRequest);
```



Volley



Volley

1. Define your tag and add it to your requests.

```
public static final String TAG = "MyTag";  
StringRequest stringRequest; // Assume this exists.  
RequestQueue mRequestQueue; // Assume this exists.  
  
// Set the tag on the request.  
stringRequest.setTag(TAG);  
  
// Add the request to the RequestQueue.  
mRequestQueue.add(stringRequest);
```

2. In your activity's `onStop()` method, cancel all requests that have this tag.

```
@Override  
protected void onStop () {  
    super.onStop();  
    if (mRequestQueue != null) {  
        mRequestQueue.cancelAll(TAG);  
    }  
}
```



Volley

```
RequestQueue mRequestQueue;

// Instantiate the cache
Cache cache = new DiskBasedCache(getCacheDir(), 1024 * 1024); // 1MB cap

// Set up the network to use HttpURLConnection as the HTTP client.
Network network = new BasicNetwork(new HurlStack());

// Instantiate the RequestQueue with the cache and network.
mRequestQueue = new RequestQueue(cache, network);

// Start the queue
mRequestQueue.start();

String url = "http://www.example.com";

// Formulate the request and handle the response.
StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
    new Response.Listener<String>() {
        @Override
        public void onResponse(String response) {
            // Do something with the response
        }
    },
    new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            // Handle error
        }
    });

// Add the request to the RequestQueue.
mRequestQueue.add(stringRequest);
```



Volley

```
ImageView mImageView;  
String url = "http://i.imgur.com/7spzG.png";  
mImageView = (ImageView) findViewById(R.id.myImage);  
...  
  
// Retrieves an image specified by the URL, displays it in the UI.  
ImageRequest request = new ImageRequest(url,  
    new Response.Listener<Bitmap>() {  
        @Override  
        public void onResponse(Bitmap bitmap) {  
            mImageView.setImageBitmap(bitmap);  
        }  
    }, 0, 0, null,  
    new Response.ErrorListener() {  
        public void onErrorResponse(VolleyError error) {  
            mImageView.setImageResource(R.drawable.image_load_error);  
        }  
    }  
));  
  
// Access the RequestQueue through your singleton class.  
MySingleton.getInstance(this).addToRequestQueue(request);
```



Volley

```
<com.android.volley.toolbox.NetworkImageView  
    android:id="@+id/networkImageView"  
    android:layout_width="150dp"  
    android:layout_height="170dp"  
    android:layout_centerHorizontal="true" />
```

```
ImageLoader mImageLoader;  
NetworkImageView mNetworkImageView;  
private static final String IMAGE_URL =  
    "http://developer.android.com/images/training/system-ui.png";  
...  
  
// Get the NetworkImageView that will display the image.  
mNetworkImageView = (NetworkImageView) findViewById(R.id.networkImageView);  
  
// Get the ImageLoader through your singleton class.  
mImageLoader = MySingleton.getInstance(this).getImageLoader();  
  
// Set the URL of the image that should be loaded into this view, and  
// specify the ImageLoader that will be used to make the request.  
mNetworkImageView.setImageUrl(IMAGE_URL, mImageLoader);
```



Volley

```
TextView mTxtDisplay;
ImageView mImageView;
mTxtDisplay = (TextView) findViewById(R.id.txtDisplay);
String url = "http://my-json-feed";

JsonObjectRequest jsonObjRequest = new JsonObjectRequest
    (Request.Method.GET, url, null, new Response.Listener<JSONObject>() {

        @Override
        public void onResponse(JSONObject response) {
            mTxtDisplay.setText("Response: " + response.toString());
        }
    }, new Response.ErrorListener() {

        @Override
        public void onErrorResponse(VolleyError error) {
            // TODO Auto-generated method stub

        }
    });

// Access the RequestQueue through your singleton class.
MySingleton.getInstance(this).addToRequestQueue(jsonObjRequest);
```



Volley

```
final String URL = "/volley/resource/12";  
// Post params to be sent to the server  
HashMap<String, String> params = new HashMap<String, String>();  
params.put("token", "AbCdEfGh123456");  
  
JsonObjectRequest req = new JsonObjectRequest(URL, new JSONObject(params),  
    new Response.Listener<JSONObject>() {  
        @Override  
        public void onResponse(JSONObject response) {  
            try {  
                VolleyLog.v("Response:%n %s", response.toString(4));  
            } catch (JSONException e) {  
                e.printStackTrace();  
            }  
        }  
    }, new Response.ErrorListener() {  
        @Override  
        public void onErrorResponse(VolleyError error) {  
            VolleyLog.e("Error: ", error.getMessage());  
        }  
    }  
));  
  
// add the request object to the queue to be executed  
ApplicationController.getInstance().addToRequestQueue(req);
```



Volley

```
final String URL = "/volley/resource/all?count=20";
JSONArrayRequest req = new JSONArrayRequest(URL, new Response.Listener<JSONArray> () {
    @Override
    public void onResponse(JSONArray response) {
        try {
            VolleyLog.v("Response:%n %s", response.toString(4));
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
        VolleyLog.e("Error: ", error.getMessage());
    }
});

// add the request object to the queue to be executed
ApplicationController.getInstance().addToRequestQueue(req);
```

Ejercicios

1. Crear una base de datos para almacenar los libros
2. Crear una copia de seguridad de la base de datos
usando un servicio Restful

<https://www.getpostman.com/collections/9a59abdbc1fcd3e0b417>



Ejercicios

<https://www.getpostman.com/collections/9a59abdbc1fcd3e0b417>

The screenshot shows a REST client interface with the following components:

- Method and URL:** A dropdown menu set to **POST** and a text field containing the URL `http://biblioteca.juanjofp.com/login`.
- Tabs:** A row of tabs for **Authorization**, **Headers (1)**, **Body** (which is selected and highlighted with an orange underline), **Pre-request Script**, and **Tests**.
- Body Type Selection:** A row of radio buttons for **form-data**, **x-www-form-urlencoded**, **raw** (which is selected), **binary**, and a dropdown menu set to **JSON (application/json)**.
- Body Content:** A text area containing a JSON object:

```
{  
  "username": "juanjo",  
  "password": "123456"  
}
```

 The text is color-coded: curly braces are grey, strings are blue, and the property names are purple.



Ejercicios

<https://www.getpostman.com/collections/9a59abdbc1fcd3e0b417>



Ejercicios

<https://www.getpostman.com/collections/9a59abdbc1fcd3e0b417>

Delete Book

DELETE ▾

http://biblioteca.juanjofp.com/book/57fc9896c5e4530e2c678faf

Authorization

Headers (2)

Body

Pre-request Script

Tests

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

JSON (application/json) ▾

1



Ejercicios

<https://www.getpostman.com/collections/9a59abdbc1fcd3e0b417>

Get Books

GET ▾

http://biblioteca.juanjofp.com/book

Authorization

Headers (2)

Body

Pre-request Script

Tests

✓	Content-Type	application/json
✓	Authorization	Bearer eyJhbGciOiJIUzI1N
	key	value

