Backgrounding

Joe Rogers
Android 310

Definition

- In Android, background operations process work on behalf of the app without an active User Interface.
- Most processing by default runs on “main thread” which also processes UI events.
- To truly run in the background, a secondary thread is needed to process the work.

Broadcast Reciever

Why you would use it?

- For backgrounding, typically used to listen for system broadcasts
  - Network Connectivity Changes
  - Battery/Power Broadcasts
  - Boot Completed
  - Many, many more.

Why you would use it cont...

- Communicating between app components.
  - Service to Activity/Fragment
  - Communicating success/failure or internal events to the user via the UI.
  - An alternative to passing a Messenger to the Service. It also allows the Activity/Fragment to go away while service is running.
  - If do this, use the LocalBroadcastManager to avoid overhead of the standard Broadcast Manager.

Receiver Pros/Cons

- Pros:
  - Lightweight, good for quick operations
  - Can do async, but not obvious, or recommended
- Cons:
  - Must complete quickly (few seconds).
  - No ability to handle other events.
  - When return from onReceive, the OS is free to kill the process.
  - Best to forward to Service for long running ops

Classes

- BroadcastReceiver
  - Standard class, must implement onReceive
- WakefulBroadcastReceiver
  - Designed to hand work to service and prevent device from sleeping while being processed.
  - Automatically establishes wakelock via startWakefulService()
  - Service should invoke completeWakefulIntent() when work is complete to release wakelock

Configure via Manifest

- Primarily used for system broadcasts
  - Add intent filters for each broadcast action to handle
  - Should not be exported if only handling system broadcasts.
- Can be used to communicate between different apps.
  - Should require a permission to access
  - Will need to be exported.

Configure via Manifest cont.

- Can dynamically enable/disable the receiver
  - Use package manager to toggle enable/disable the broadcast receiver.
  - Handy if only need to listen for broadcasts in certain situations
  - For example, if detect no connectivity, enable receiver to be notified when connectivity is restored.

Configure as needed

- Best for communications between app components.
  - Enable onResume() and disable during onPause()
  - Fragment/Activity typically monitors to present information to the user via the UI.
  - Should use LocalBroadcastManager to register receiver and send broadcasts to limit overhead.

Service

Why you would use it?

- Handy for long running tasks where the code does not require access to the UI.
- Another alternative is to provide access to other applications (not common).
  - Should require a permission and exported

Service Pros/Cons

- Pros
  - Best designed for long running operations
  - Can be used to handle callbacks
  - Interact via Intents, or can expose a bindable interface, or both at same time.
- Cons
  - Still requires a background thread in most cases
  - Can consume resources/memory if not shut down
  - No built in support for response communication

Classes

- Service
  - Basic class for all services. Typically what is implemented if you are implementing a custom service using intents or bindings.
- IntentService
  - Specialized service for processing intents. The service also automatically sets up its own background thread. Quickest, easiest service to use.

Starting via Intents

- Uses standard intents to start the service
- onStartCommand()
  - Primary entry point for each intent
  - startId will be different for each intent
  - flags indicates if service was restarted.
- Service runs until either Context.stopService() or stopSelf() is called. Calling stopSelf(startId) will keep service running, if startId, is not last id sent to keep processing the additional intents.

OnStartCommand responses

- START_NOT_STICKY
  - Service will not remain running if the OS needs to kill it. Great for jobs that run on a periodic basis.
- START_REDELIVER_INTENT
  - OS will redeliver last intent if the OS had to kill service. Useful for jobs that must finish in-order.
- START_STICKY
  - OS restarts service if had to be destroyed. Intent may be null. Useful for playing music, etc.

Binding a service

- Binding provides a remote interface to the service for direct interaction by activity/frag.
- Most apps would use a local direct interface to interact (easy).
- If need to cross processes or apps, must use either a messenger (intermediate) or Android Interface Definition Language aka AIDL (advanced)

Steps to bind directly

- Define an interface to share between service and activities.
- The interface may be implemented by the service.
- The service creates a public instance of the Binder class which should add a method to return the interface.

Steps to bind directly cont....

- The activity or fragment should invoke onBind() during onStart(). It will pass a service connection callback.
- When onServiceConnected() cast the binder to the class and get a reference to the interface.
- Now activity or fragment may invoke the methods on the interface.

Steps to bind directly cont....

- The activity or fragment should release connection to service via unbindService() during onStop().

Wake Lock

What is a WakeLock

- Specialized class used to keep the device awake even when the user is not interacting with the screen.
- Required for certain operations
  - Playing audio in the background with screen off
  - Keeping device awake to process background work when notified via system broadcast, alarm, etc.
- **Not** required using job scheduler.
- Ensure to release wake lock as soon as possible to avoid keeping device awake unnecessarily.

Levels of WakeLocks

- PARTIAL_WAKE_LOCK
  - Keeps cpu awake to process work even if user presses power button
  - Screen is allowed to go off
  - Primary wake lock you will use.
- PROXIMITY_SCREEN_OFF_WAKE_LOCK
  - Turns off/on the screen if proximity sensor detects object or not. Think talking on the phone. New in Lollipop, to make dialers easier.

Note on keeping screen on

- If you want to keep the screen on, should use the WindowManager attribute FLAG_KEEP_SCREEN_ON.
  - Requires no special permission.
  - Useful if showing a barcode for scanning.
  - Can only be done in an Activity.
  - getWindow().addFlag() to turn on. Use getWindow().clearFlag() to remove requirement.

Getting a wakelock

- Add wake lock permission to manifest.
- Get reference to the PowerManager system service.
- Use newWakeLock to build your wakelock.
- Call acquire() to activate wake lock
- Do work that needs to keep device awake
  - Wrap in try/finally
- Call release() to allow cpu to sleep.
  - Best to call in finally block to ensure wake lock is release even if exception occurs.

AsyncTask

What is it?

- The easiest go to way to run something quickly in the background.
- Essentially, implement doInBackground() to perform the background work.
- Implement onPostExecute() to process any result in the “main” UI thread.
- Has both a serialized and parallel thread pool.

AsyncTask Pros/Cons

- Pros
  - Easy to use, implements most of the hard work.
  - Great for short operations.
  - Default is serialize operations, but easy to run in parallel by specifying the proper executor.
- Cons
  - Requires manual tracking, to ensure canceled
  - Easy to leak UI element
  - Create own thread pool for long operations.

Handlers, Loopers, etc

What is a Handler

- Essentially a message queue that can be used to queue up work in the main thread (default) or another thread.
- Useful to transfer work back to main thread if in an asynchronous callback.
- Also can be used delay work for short periods of time, or configure short time outs.

What is a Handler Thread or Looper

- Handler Thread
  - A specialized thread that builds a Looper than may be used in a Handler to processes messages, runnables in another thread.
- Looper
  - Runs the message loop for a thread.
  - Handler with default constructor will automatically use the looper of the main thread.

Demo Notes

Demo locations

- WakefulBroadcastReciever
  - See AlarmBroadcastReceiver and NetworkAlarmIntentService
- BroadcastReceiver (system broadcast) as well as dynamically enabling/disabling receiver
  - See ConnectivityBroadcastReceiver
- LocalBroadcastManager and dynamic broadcasts
  - See NetworkStatusBroadcastReceiver, IncidentFragment

Demo locations cont.

- Bound Service
  - See LocationService, MainActivity
- WakeLock
  - See NetworkIntentService
- AsyncTask
  - See NetworkAsyncJobService
- Hander, HandlerThread, Looper
  - See NetworkHandlerJobService

Resources

Resources

- Broadcast Receiver
- Wakeful Broadcast Receiver
- Service JavaDoc (with examples)
- Service Guide
- AIDL (Android Interface Device Language)
- Wakelocks and Keeping screen on

Resources Cont.

- AsyncTask
- Handler
- HandlerThread
- Looper
- Custom Threading (Advanced)