Broadcast Receiver

CS43 - Mobile Applications
Instructor: Bo Sheng

Overview (I)

- A broadcast receiver listens for relevant broadcast messages to trigger an event, e.g.,
  - The camera button was pressed.
  - The battery is low.
  - A new application was installed.
- A user-generated component can also send a broadcast.
  - A calculation was finished.
  - A particular thread has started.
- All broadcast receivers extend the abstract class BroadcastReceiver or one of its subclasses.

Overview (II)

- Two ways to register an instance of BroadcastReceiver
  - Dynamically register with Context.registerReceiver()
  - Statically publish through <receiver> in AndroidManifest.xml.
- If registering a receiver in Activity.onResume(), you should unregister it in Activity.onPause().
  - You won’t receive intents when paused.
- Do not unregister in Activity.onSaveInstanceState().
  - This won’t be called if the user moves back in the history stack.

Two Classes of Broadcasts

- Normal broadcasts
  - Sent with Context.sendBroadcast
  - Completely asynchronous.
  - All receivers are run in an undefined order, often at the same time.
  - More efficient, but receivers cannot return results or abort broadcast.
- Ordered broadcasts
  - Sent with Context.sendOrderedBroadcast
  - Delivered to one receiver at a time.
  - As each receiver executes in turn, it can propagate a result to the next receiver, or it can completely abort the broadcast.
  - The order can be controlled with the android:priority attribute of the matching intent-filter; receivers with the same priority will be run in an arbitrary order.

Intents for broadcast

- The Intent class is used for sending and receiving broadcasts.
- The Intent broadcast mechanism is completely separate from Intents that are used to start Activities with Context.startActivity().
  - A BroadcastReceiver cannot see or capture Intents used with startActivity();
  - When you broadcast an Intent, you will never find or start an Activity.
- The two operations are semantically very different
  - Starting an Activity with an Intent is a foreground operation that modifies what the user is currently interacting with;
  - Broadcasting an Intent is a background operation that the user is not normally aware of.

Standard Broadcast Actions

- Current standard actions that Intent defines for receiving broadcasts (usually through registerReceiver(BroadcastReceiver, IntentFilter) or a <receiver> tag in a manifest)
  - ACTION_TIME_TICK
  - ACTION_TIME_CHANGED
  - ACTION_TIMEZONE_CHANGED
  - ACTION_BOOT_COMPLETED
  - ACTION_PACKAGE_ADDED
  - ACTION_PACKAGE_CHANGED
  - ACTION_PACKAGE_REMOVED
  - ACTION_PACKAGE_RESTARTED
  - ACTION_PACKAGE_DATA_CLEARED
  - ACTION_UID_REMOVED
  - ACTION_BATTERY_CHANGED
  - ACTION_POWER_CONNECTED
  - ACTION_POWER_DISCONNECTED
  - ACTION_SHUTDOWN
Receiver Lifecycle

- A BroadcastReceiver object is only valid for the duration of the call to `onReceive(Context, Intent)`.
  - Once returns, the object is considered to be finished and no longer active.

- Implication for `onReceive(Context, Intent)`: don’t perform asynchronous operation (10 seconds limit)
  - You will need to return from the function to handle the asynchronous operation
  - However, at that point the BroadcastReceiver is no longer active and thus the system is free to kill its process before the asynchronous operation completes.

- Don’t show a dialog or bind to a service from within a BroadcastReceiver.
  - For the former, you should instead use the `NotificationManager` API.
  - For the latter, you can use `Context.startService()` to send a command to the service.

Process Lifecycle

- A process running `onReceive(Context, Intent)` is considered to be a foreground process and will be kept running by the system except under cases of extreme memory pressure.

- Once return from `onReceive()`, the BroadcastReceiver is no longer active, and its hosting process is only as important as any other application components that are running in it.

- If a process is only hosting a BroadcastReceiver, then upon returning the system will consider the process to be empty and aggressively kill it.

- For long-running operations, it is often to use a `Service` in conjunction with a BroadcastReceiver to keep the containing process active for the entire time of the operation.

An Example of Broadcast Receiver

- Shows how to start a service based on a broadcasted event, such as when the camera button is pressed.

- The broadcast receiver is used to listen for the specified event(s) and subsequently launch the service.

- The activity sets up a broadcast receiver and sets up an intent with the filter for the camera button.

- Then, the broadcast receiver is started and this intent filter is passed to it using the `registerReceiver()` method.

MainActivity.java

- The broadcast receiver (SimpleReceiver) responds to `ACTION_BOOT_COMPLETED` and `ACTION_PACKAGE_ADDED`.

SimpleReceiver.java

- If `ACTION_BOOT_COMPLETED` broadcast intent is received by SimpleReceiver, the SimpleService will be started.

SimpleService.java

- SimpleService is a toy service. It will show a toast message when being started.
SimpleService.java

- A toast provides simple feedback about an operation in a small popup.
- If user response to a status message is required, consider instead using a Notification.

Running the Example

- We can use "adb shell am broadcast" command to send a broadcast message to SimpleReceiver in the emulator.
  - Using activity manager (am) for debugging.

  - The command
    - adb.exe shell am broadcast
    - a android.intent.action.BOOT_COMPLETED
    - c android.intent.category.HOME
    - n edu.umb.cs443.simplereceiver/.SimpleReceiver


SimpleReceiver Screenshot

SimpleService is running

- Settings → App → Running

Settings → Developer options → Running services

- Enable Developer options
  - Setting → About the device
  - Press ‘kernel/build version’ 7 times
Custom Intents and Broadcasting

• We can create a custom broadcast intent.
• This example has two receivers.
• The first receiver OutgoingReceiver captures the system broadcast intent "android.intent.action.PHONE_STATE" and sends a custom broadcast intent "edu.umb.cs443.intent.action.TEST".
  – Log "HIT OUTGOING" assertion.
• The second receiver IncomingReceiver captures the custom broadcast intent.
  – Log "GOT THE INTENT" assertion.
• Use "extended controls" to simulate an incoming phone call to trigger "android.intent.action.PHONE_STATE".

AndroidManifest

- Need to use permission READ_PHONE_STATE.

```xml
<application>
  <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
  <application android:allowBackup="true" android:icon="@drawable/ic_launcher" android:label="@string/app_name" android:theme="@style/AppTheme">
    <receiver android:name="edu.umb.cs443.customreceiver.OutgoingReceiver" android:enabled="true" android:exported="true">
      <intent-filter>
        <action android:name="android.intent.action.PHONE_STATE"/>
      </intent-filter>
    </receiver>
    <receiver android:name="edu.umb.cs443.customreceiver.IncomingReceiver" android:enabled="true" android:exported="false">
      <intent-filter>
        <action android:name="edu.umb.cs443.intent.action.TEST"/>
      </intent-filter>
    </receiver>
    <activity android:name="edu.umb.cs443.customreceiver.MainActivity" android:label="@string/title_activity_main" android:name="android.intent.category.LAUNCHER">
      <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
      </intent-filter>
    </activity>
  </application>
</application>
```