Введение в разработку под Android

Семинар 10.

Макаров И.О. 20.11.2020

На этом семинаре

- Broadcast’ы
- Много теории и мало кода
- Полезные ссылки

Broadcasts Overview

https://developer.android.com/guide/components/broadcasts

Android apps can send or receive broadcast messages from the Android system and other Android apps, similar to the publish-subscribe design pattern. These broadcasts are sent when an event of interest occurs.

For example, the Android system sends broadcasts when various system events occur, such as when the system boots up or the device starts charging. Apps can also send custom broadcasts, for example, to notify other apps of something that they might be interested in (for example, some new data has been downloaded).

Apps can register to receive specific broadcasts. When a broadcast is sent, the system automatically routes broadcasts to apps that have subscribed to receive that particular type of broadcast.

About system broadcasts

https://developer.android.com/guide/components/broadcasts

The system automatically sends broadcasts when various system events occur, such as when the system switches in and out of airplane mode. System broadcasts are sent to all apps that are subscribed to receive the event.

The broadcast message itself is wrapped in an `Intent` object whose action string identifies the event that occurred (for example `android.intent.action.AIRPLANE_MODE`). The intent may also include additional information bundled into its extra field. For example, the airplane mode intent includes a boolean extra that indicates whether or not Airplane Mode is on.

For more information about how to read intents and get the action string from an intent, see Intents and Intent Filters.

For a complete list of system broadcast actions, see the `BROADCAST_ACTIONS.TXT` file in the Android SDK. Each broadcast action has a constant field associated with it. For example, the value of the constant `ACTION_AIRPLANE_MODE_CHANGED` is `android.intent.action.AIRPLANE_MODE`. Documentation for each broadcast action is available in its associated constant field.

Changes to system broadcasts

https://developer.android.com/guide/components/broadcasts

As the Android platform evolves, it periodically changes how system broadcasts behave. Keep the following changes in mind if your app targets Android 7.0 (API level 24) or higher, or if it's installed on devices running Android 7.0 or higher.

https://developer.android.com/guide/components/broadcasts#changes-system-broadcasts

If your app targets API level 26 or higher, you cannot use the manifest to declare a receiver for *implicit* broadcasts (broadcasts that do not target your app specifically), except for a few implicit broadcasts that are exempted from that restriction. In most cases, you can use scheduled jobs instead.

Manifest-declared receivers

https://developer.android.com/guide/components/broadcasts

If your Target android version is more than Android O. if you declare receivers in manifest they wont work. So you need to register inside your activity.

If your app targets API level 26 or higher, you cannot use the manifest to declare a receiver for implicit broadcasts (broadcasts that do not target your app specifically), except for a few implicit broadcasts that are exempted from that restriction. In most cases, you can use scheduled jobs instead

```
<receiver android:name=".MyBroadcastReceiver"  android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
        <action android:name="android.intent.action.INPUT_METHOD_CHANGED" />
    </intent-filter>
</receiver>
```

```
public class MyBroadcastReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            ...
        }
    }
```

Effects on process state

https://developer.android.com/guide/components/broadcasts

The state of your BroadcastReceiver (whether it is running or not) affects the state of its containing process, which can in turn affect its likelihood of being killed by the system. For example, when a process executes a receiver (that is, currently running the code in its onReceive() method), it is considered to be a foreground process. The system keeps the process running except under cases of extreme memory pressure.

However, once your code returns from onReceive(), the BroadcastReceiver is no longer active. The receiver's host process becomes only as important as the other app components that are running in it. If that process hosts only a manifest-declared receiver (a common case for apps that the user has never or not recently interacted with), then upon returning from onReceive(), the system considers its process to be a low-priority process and may kill it to make resources available for other more important processes.

For this reason, you should not start long running background threads from a broadcast receiver. After onReceive(), the system can kill the process at any time to reclaim memory, and in doing so, it terminates the spawned thread running in the process. To avoid this, you should either call goAsync() (if you want a little more time to process the broadcast in a background thread) or schedule a JobService from the receiver using the JobScheduler, so the system knows that the process continues to perform active work.

For more information, see Processes and Application Life Cycle.

Effects on process state

https://developer.android.com/guide/components/broadcasts

```
public class MyBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        final PendingResult pendingResult = goAsync();
        Task asyncTask = new Task(pendingResult, intent);
        asyncTask.execute();
    }

    private static class Task extends AsyncTask<String, Integer, String> {

        private final PendingResult pendingResult;
        private final Intent intent;

        private Task(PendingResult pendingResult, Intent intent) {
            this.pendingResult = pendingResult;
            this.intent = intent;
        }

        @Override
        protected String doInBackground(String... strings) {
            return intent.getAction();
        }

        @Override
        protected void onPostExecute(String s) {
            super.onPostExecute(s);
            pendingResult.finish();          // Must call finish() so the BroadcastReceiver can be recycled.
        }
    }
}
```

Sending broadcasts

https://developer.android.com/guide/components/broadcasts

Android provides three ways for apps to send broadcast:

- The sendOrderedBroadcast(Intent, String) method sends broadcasts 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 so that it won't be passed to other receivers. The order receivers run in 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.

- The sendBroadcast(Intent) method sends broadcasts to all receivers in an undefined order. This is called a Normal Broadcast. This is more efficient, but means that receivers cannot read results from other receivers, propagate data received from the broadcast, or abort the broadcast.

- ~~The LocalBroadcastManager.sendBroadcast method sends broadcasts to receivers that are in the same app as the sender. If you don't need to send broadcasts across apps, use local broadcasts. The implementation is much more efficient (no interprocess communication needed) and you don't need to worry about any security issues related to other apps being able to receive or send your broadcasts.~~

Sending broadcasts

https://developer.android.com/guide/components/broadcasts

Android provides three ways for apps to send broadcast:

- The sendOrderedBroadcast(Intent, String) method sends broadcasts 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 so that it won't be passed to other receivers. The order receivers run in 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.

- The sendBroadcast(Intent) method sends broadcasts to all receivers in an undefined order. This is called a Normal Broadcast. This is more efficient, but means that receivers cannot read results from other receivers, propagate data received from the broadcast, or abort the broadcast.

- ~~The LocalBroadcastManager.sendBroadcast method sends broadcasts to receivers that are in the same app as the sender. If you don't need to send broadcasts across apps, use local broadcasts. The implementation is much more efficient (no interprocess communication needed) and you don't need to worry about any security issues related to other apps being able to receive or send your broadcasts.~~

Note: Although intents are used for both sending broadcasts and starting activities with **startActivity(Intent)**, these actions are completely unrelated. Broadcast receivers can't see or capture intents used to start an activity; likewise, when you broadcast an intent, you can't find or start an activity.

Sending with permissions

https://developer.android.com/guide/components/broadcasts

When you call sendBroadcast(Intent, String) or sendOrderedBroadcast(Intent, String, BroadcastReceiver, Handler, int, String, Bundle), you can specify a permission parameter. Only receivers who have requested that permission with the tag in their manifest (and subsequently been granted the permission if it is dangerous) can receive the broadcast. For example, the following code sends a broadcast:

```
sendBroadcast(new Intent("com.example.NOTIFY"),
        Manifest.permission.SEND_SMS);
```

To receive the broadcast, the receiving app must request the permission as shown below:

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

You can specify either an existing system permission like SEND_SMS or define a custom permission with the <permission> element. For information on permissions and security in general, see the System Permissions.

Note: Custom permissions are registered when the app is installed. The app that defines the custom permission must be installed before the app that uses it.

Receiving with permissions

https://developer.android.com/guide/components/broadcasts

If you specify a permission parameter when registering a broadcast receiver (either with registerReceiver(BroadcastReceiver, IntentFilter, String, Handler) or in <receiver> tag in your manifest), then only broadcasters who have requested the permission with the <uses-permission> tag in their manifest (and subsequently been granted the permission if it is dangerous) can send an Intent to the receiver.
For example, assume your receiving app has a manifest-declared receiver as shown below:

```
<receiver android:name=".MyBroadcastReceiver"
          android:permission="android.permission.SEND_SMS">
    <intent-filter>
        <action android:name="android.intent.action.AIRPLANE_MODE"/>
    </intent-filter>
</receiver>
```

Or your receiving app has a context-registered receiver as shown below:

```
IntentFilter filter = new IntentFilter(Intent.ACTION_AIRPLANE_MODE_CHANGED);
registerReceiver(receiver, filter, Manifest.permission.SEND_SMS, null );
```

Дополнительно можно

https://developer.android.com/courses/fundamentals-training

- "Unit 3: Working in the background" -> "Lesson 7: Background tasks" -> "7.3: Broadcast receivers”
- security-and-best-practices
- broadcast-exceptions
- Background Optimizations (Android Development Patterns S3 Ep 14)