Android Developer Fundamentals

Background Tasks

Lesson 7

7.3 Broadcast Receivers

Contents

- Broadcast intents
- Broadcast receivers
- Implementing broadcast receivers
- Custom broadcasts
- Security
- Local broadcasts

Broadcast Intents

Broadcast vs. Activity

Use implicit intents to send broadcasts or start activities

Sending broadcasts

- Use sendBroadcast()
- Can be received by any application registered for the intent
- Used to notify all apps of an event

Starting activities

- Use startActivity()
- Find a single activity to accomplish a task
- Accomplish a specific action

Broadcast Receivers

What is a broadcast receiver?

- Listens for incoming intents sent by sendBroadcast()
  - In the background
- Intents can be sent
  - By the system, when an event occurs that might change the behavior of an app
  - By another application, including your own

Broadcast receiver always responds

- Responds even when your app is closed
- Independent from any activity
- When a broadcast intent is received and delivered to onReceive(), it has 5 seconds to execute, and then the receiver is destroyed

System broadcasts

- Automatically delivered when certain events occur
- After the system completes a boot
  - `android.intent.action.BOOT_COMPLETED`
- When the wifi state changes
  - `android.net.wifi.WIFI_STATE_CHANGED`

Custom broadcasts

- Deliver any custom intent as a broadcast
  - `sendBroadcast()` method—asynchronous
  - `sendOrderedBroadcast()`—synchronously
  - `android.example.com.CUSTOM_ACTION`

sendBroadcast()

- All receivers of the broadcast are run in an undefined order
- Can be at the same time
- Efficient
- Use to send custom broadcasts

sendOrderedBroadcast()

- Delivered to one receiver at a time
- Receiver can propagate result to the next receiver or abort the broadcast
- Control order with android:priority of matching intent filter
- Receivers with same priority run in arbitrary order

Implementing Broadcast Receivers

Steps for creating a broadcast receiver

1. Subclass `BroadcastReceiver`
2. Implement `onReceive()` method
3. Register to receive broadcast
    - Statically, in AndroidManifest
    - Dynamically, with `registerReceiver()`

File > New > Other > BroadcastReceiver

```
public class CustomReceiver extends BroadcastReceiver {
    public CustomReceiver() {
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        // TODO: This method is called when the BroadcastReceiver
        //  is receiving an Intent broadcast.
        throw new UnsupportedOperationException("Not yet
implemented");
    }
}
```

Register in Android Manifest

- <receiver> element inside <application>
- <intent-filter> registers receiver for specific intents

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

Register dynamically

- In onCreate() or onResume()
- Use registerReceiver() and pass in the intent filter
- Must unregister in onDestroy() or onPause()

```
registerReceiver(mReceiver, mIntentFilter)
unregisterReceiver(mReceiver)
```

Available intents

- ACTION_TIME_TICK
- ACTION_TIME_CHANGED
- ACTION_TIMEZONE_CHANGED
- ACTION_BOOT_COMPLETED
- ACTION_PACKAGE_ADDED
- ACTION_PACKAGE_CHANGED
- ACTION_PACKAGE_REMOVED
- ACTION_PACKAGE_RESTART
- ACTION_PACKAGE_DATA_CLEARED
- ACTION_PACKAGES_SUSPENDED
- ACTION_PACKAGES_UNSUSPENDED
- ACTION_UID_REMOVED
- ACTION_BATTERY_CHANGED
- ACTION_POWER_CONNECTED
- ACTION_POWER_DISCONNECTED

Implement onReceive()

```
@Override
public void onReceive(Context context, Intent intent) {
    String intentAction = intent.getAction();
    switch (intentAction){
        case Intent.ACTION_POWER_CONNECTED:
            break;
        case Intent.ACTION_POWER_DISCONNECTED:
            break;
    }
}
```

Custom Broadcasts

Custom broadcasts

- Sender and receiver must agree on unique name for intent (action name)
- Define in activity and broadcast receiver

```
private static final String ACTION_CUSTOM_BROADCAST =

"com.example.android.powerreceiver.ACTION_CUSTOM_BROADCAST";
```

Send custom broadcasts

```
Intent customBroadcastIntent =
    new Intent(ACTION_CUSTOM_BROADCAST);

LocalBroadcastManager.getInstance(this)
    .sendBroadcast(customBroadcastIntent);
```

Destroy!

```
@Override
protected void onDestroy() {
    LocalBroadcastManager.getInstance(this)
            .unregisterReceiver(mReceiver);
    super.onDestroy();
}
```

Security

Security

- Receivers cross app boundaries
- Make sure namespace for intent is unique and you own it
- Other apps can send broadcasts to your receiver—use permissions to control this
- Other apps can respond to broadcast your app sends
- Access permissions can be enforced by sender or receiver

Controlling permission sender

- `void sendBroadcast (Intent intent, String receiverPermission)`

- Receivers must request permission with `<uses-permission>` in AndroidManifest.xml

Controlling permission receiver

- `registerReceiver(BroadcastReceiver, IntentFilter, String, android.os.Handler)`
- or in `<receiver>` tag

- Senders must request permission with `<uses-permission>` in AndroidManifest.xml

Local Broadcast Manager

Local Broadcast Manager

- For broadcasts only in your app
- No security issues since no cross-app communication

```
LocalBroadcastManager.sendBroadcast()
LocalBroadcastManager.registerReceiver()
```

Register local broadcast manager

```
LocalBroadcastManager.getInstance(this)
   .registerReceiver( mReceiver,
         new
IntentFilter(ACTION_CUSTOM_BROADCAST));
```

Learn more

- BroadcastReceiver Reference
- Intents and Intent Filters Guide
- LocalBroadcastManager Reference
- Manipulating Broadcast Receivers On Demand

What's Next?

- Concept Chapter: 7.3 C Broadcast Receivers
- Practical: 7.3 P Broadcast Receivers

END