Android register sms receiver programmatically

Android apps can send or receive broadcast messages from Android and other Android apps, similar to the publishing subscription design template. These broadcasts are sent when an event of interest occurs. For example, Android sends broadcasts when various system events occur, such as when the system loads or the device starts charging. Apps can also send user broadcasts, for example, to notify other apps of something they might be interested in (for example, some new data has been downloaded). Apps can register for specific broadcasts. When you send a broadcast, the system automatically sends broadcasts to apps that subscribe to a certain type of broadcast. Generally speaking, broadcasts can be used as a messaging system in apps and beyond the usual user stream. However, you should be careful not to overuse the ability to respond to broadcasts and run tasks in the background that can contribute to the slow performance of the system as described in the following video. The system automatically sends transmissions at various system events, such as when the system is switched on and off the plane mode. System broadcasts are sent to all apps that subscribe to the event. The broadcast message itself is wrapped in an Intent object whose action line determines the event that occurred (such as android.intent.action.AIRPLANE_MODE). Intention may also include additional information bundled into your additional field. For example, the intention of the airplane mode includes a boolean additionally, which indicates whether the plane mode is enabled. For more information on how to read intentions and get a string of actions from intentions, see intentions and intention filters. The full list of system broadcasting activities can be viewed on BROADCAST_ACTIONS. TXT file in Android SDK. Each transfer action has a permanent field associated with it. For example, the value of a permanent ACTION_AIRPLANE_MODE_CHANGED android.intent.action.AIRPLANE_MODE. Documentation for each broadcast is available in a related permanent field. Changes in system broadcasts as the Android platform evolves periodically changes the behavior of system broadcasts. Keep in mind the following changes if your app is aimed at Android 7.0 (API level 24) or higher, or if it's installed on devices running Android 7.0 or higher. Android 9 Starting with Android 9 (API level 28), NETWORK_STATE_CHANGED_ACTION broadcasting does not receive information about the user's location or personal data. Also, if your app is installed on a device running Android 9 or higher, system Wi-Fi does not contain SSID, BSSID, connection information or scan results. For this information, call getConnectionInfo instead. Android 8.0 Starting with Android 8.0 (API level 26), the system imposes additional restrictions on Receivers. If your app is aimed at Android 8.0 or higher, you can't use the manifest to announce the receiver for most implicit broadcasts (broadcasts that aren't specifically targeted at your app). You can still use a context-registered receiver when the user is actively using your app. Android 7.0 Android 7.0 (API level 24) and above do not send the following system broadcasts: ACTION_NEW_PICTURE ACTION_NEW_VIDEO In addition, apps focused on Android 7.0 and above must register the CONNECTIVITY_ACTION broadcast using registerReceiver (BroadcastReceiver, IntentFilter). The receiver's announcement in the manifest doesn't work. Receiving app broadcasts can be broadcast in two ways: through explicitly stated receivers and context-registered receivers. Manifesto-declared receivers If you announce a broadcast receiver in your manifest, the system launches your app (if the app is not yet running) when the broadcast is sent. Note: If your app is aimed at API level 26 or higher, you cannot use the manifest to announce a receiver for implicit broadcasts (broadcasts that are not aimed at your app specifically), except for a few implicit transmissions that are exempt from this restriction. In most cases, you can use scheduled jobs instead. To announce the broadcast receiver in the manifest, follow the following steps: specify an item in the application's manifest. Intent filters determine the actions of the broadcast to which your recipient subscribes. <receiver android:name=. MyBroadcastReceiver android:exported'true'gt; qlt'-filter'gt; qlt;action android:android.intent.action.BOOT_COMPLETED'lt;lt;action-action-android:name'android.intent.action.INPUT_METHOD_CHANGED'lt'gt; The receiver broadcasts in the following example magazines and displays the content of the broadcast: private const val TAG - MyBroadcastReceiver class MyBroadcastReceiver : BroadcastReceiver () - redefine pleasure onReceive (context: Context, Intention: Intention) - StringBuilder (). Apply app (Action: $intent.action) app (URI: $'intent.toUri (Intent.URI_INTENT_SCHEME) toString.). >. magazine, Toast.LENGTH_LONG). @Override public void on Receive (Context context, intention of intent) - StringBuilder sb - new StringBuilder sb.append sb.append (URI: - intent.toUri (Intent.URI_INTENT_SCHEME).toString () -); String magazine sb.toString(); Log.d (TAG, magazine); Toast.makeText magazine, Toast.LENGTH_LONG)... The system package manager registers the receiver when the application is installed. The receiver becomes a separate entry point to your app, which means that the system can start the app and deliver the broadcast if the app is zlt;/receiver.html currently not working. The system creates a new BroadcastReceiver component to handle every broadcast it receives. This object is only valid for the time of the onReceive call. As soon as the code returns from this method, the system considers the component no longer active. Context-registered receivers To register a receiver with context follow the following steps: Create a copy of BroadcastReceiver. val br: BroadcastReceiver - MyBroadcastReceiver () BroadcastReceiver br - new MyBroadcastReceiver (); Create IntentFilter and register the receiver by calling RegisterReceiver (ConnectivityManager.CONNECTIVITY_ACTION, IntentFilter): Val - IntentFilter filter (Intent.ACTION_AIRPLANE_MODE_CHANGED ConnectivityManager.CONNECTIVITY_ACTION). filter.addAction (Intent.ACTION_AIRPLANE_MODE_CHANGED); this.registerReceiver (br, filter); Note: To register for local broadcasts, call LocalBroadcastManager.registerReceiver (BroadcastReceiver, IntentFilter) instead. Context-registered receivers receive broadcasts as long as their registration context is valid. For example, if you register in the context of action, you get broadcasts until the action is destroyed. If you register in the context of the app, you get broadcasts as long as the app is running. To stop accepting broadcasts, call unregisteredReceiver (android.content.BroadcastReceiver). Be sure to have an unregistered receiver when you no longer need it, or the context is no longer valid. Be aware of where you register and register the receiver, for example, if you register the receiver in onCreate (Bundle) using the context of the action, you must not register it in onDestroy() to prevent the receiver from leaking out of the context of the action. If you register a receiver in onResume, you must unregistered it in onPause () to prevent its registration several times (if you do not want to receive the broadcast when suspended, and this may reduce unnecessary system overheads). Don't sign up for onSaveInstanceState (Bundle) because it's not called if the user returns to the story stack. The effect on the state of the process State of your BroadcastReceiver (whether it works or not) affects the state of its containing process, which in turn can affect the likelihood of his death by the system. For example, when the process performs the receiver (i.e. currently performing the code in its onReceive method), it is considered a foreground process. The system supports the process works, except in cases of extreme memory pressure. However, once your code returns from onReceive, BroadcastReceiver is no longer active. The process of placing the receiver becomes the same like the other components of the app that work in it. If this process only contains (a common case for applications that the user has never interacted with recently), then on return from onReceive, the system considers its process a low-priority process and can kill it to make resources available to other, more important processes. For this reason, you should not start a long-term run of background streams from the broadcast receiver. After onReceive (), the system can kill the process at any time to restore memory, and in doing so, it completes the generated flow of running in the process. To avoid this, call goAsync (if you want a little more time to handle the broadcast in the background) or schedule JobService from the recipient using JobScheduler, so that the system knows that the process continues to perform active work. For more information see the following snippet shows BroadcastReceiver, which uses the goAsync () flag, that it needs more time to finish after onReceive () is completed. This is especially useful if the work you want to complete in onReceive () is long enough to force the UI stream to skip the frame (qgt;16ms), making it better suited to the background flow. Private const val TAG - MyBroadcastReceiver Class MyBroadcastReceiver : BroadcastReceiver () - redefine fun onReceive (context: Context, Intention: Intention) - val pendingResult: PendingResult - goAsync () val asyncTask - Task (pendingResult, intent. Private val Intentions: Intention) : AsyncTask'lt'Lt;String, int, string'gt;)- override fun doInBackground (vararg params: String?): String yu Val SB - StringBuilder () sb.append (Action: $'intent.action) sb .append (URI: $'intent.Uri to (Intent.URI_INTENT_SCHEME)) return toString ()). super.onPostExecute (result) / Must cause a finish () so that BroadcastReceiver can be redesigned. pendingResult.finish () - MyBroadcastReceiver's public class expands BroadcastReceiver - Private static string TAG - MyBroadcastReiver; @Override public void onReceive (Context, Intention of Intent) - the final wait for the result and goAsync (); AsyncTask's task is a new task (pendingResult, intention); asyncTask.execute(); - Private static class challenge expands AsyncTask - private pendingResult finale in anticipation Private final intent intentions; Private task, integer, PendingResult in anticipation of the Result, intention of intent) - this.pendingResult - pendingResult; this.intent - intention; - @Override protected line doInBackground (String... StringBuilder sb - new StringBuilder sb.append sb.append (URI: - intent.toUri (Intent.URI_INTENT_SCHEME).toString () -); String magazine sb.toString(); Log.d(TAG, возврат журнала; - @Override защищенная пустота наPostExecute (String s)</String,> </String,> </String,> Must cause a finish (), so broadcastReceiver can be redesigned. pendingResult.finish(); Sending Android broadcasts provides three ways for apps to send a broadcast: sendOrderedBroadcast (Intention, String) sends broadcasts to one receiver at a time. As each receiver performs in turn, he can extend the result to the next receiver, or he may completely interrupt the transmission so that it will not be transferred to other receivers. Receiving order receivers can be controlled with android: the priority attribute of matching the intentional filter; receivers with the same priority will work at random. The sendBroadcast (Intent) method sends transmissions to all recipients indefinitely. This is called a normal broadcast. This is more effective, but means that recipients cannot read the results of other receivers, distribute data from the broadcast, or interrupt the broadcast. LocalBroadcastManager.sendBroadcast sends broadcasts to recipients who are in the same app as the sender. If you don't need to stream through apps, use local broadcasts. Implementation is much more efficient (no interprocess communication required) and you don't need to worry about any security issues associated with other apps being able to receive or send your broadcasts. The next piece of code demonstrates how to send a broadcast by creating an intention and by calling sendBroadcast. Intention ()... - intention - intent.setAction (com.example.broadcast.MY_NOTIFICATION) intention.putExtra (data, Please note me senpai!) sendBroadcast (intention) - Intention of intent and new intentions (); intent.setAction (com.example.broadcast.MY_NOTIFICATION); intent.putExtra (data, Please note me senpai!); SendBroadcast (intention); The broadcast message is wrapped in an Object of Intent. The intent action line should provide the syntax of the Java application package's name and unambiguously identify the broadcast event. You can attach more information to the intention with putExtra (String, Bundle). You can also limit the broadcast to a set of apps in the same organization by calling setPackage (String) for intent. Note: Although intentions are used to both send broadcasts and start actions with startActivity (Intent), these actions are completely unrelated. Broadcast receivers cannot see or capture the intentions used to start the action; similarly, when you broadcast the intention, you can't find or start the action. Restricting broadcasts with permission allows you to limit broadcasts to a set of apps that have certain permissions. You can impose restrictions on the sender or recipient of the broadcast. With permissionM when you call sendBroadcast (Intention, Line) or SendOred Broadcast (Intention, Line, BroadcastReceiver, Handler, Int, Line, Kit), you can specify the resolution option. Only recipients who have requested permission with a tag in their manifest (and permission has been granted if it is dangerous) can get a broadcast. For example, the following code sends a broadcast: sendBroadcast (com.example(com.example.), Manifest.permission.SEND_SMS) sendBroadcast (new intention (com.example.MANIFEST.PERMISSION.SEND_SMS); To receive the broadcast, the receiving app must request permission, as shown below: android.permission.SEND_SMS you can specify either the existing resolution of the system as SEND_SMS or determine the user resolution with the element. For information about permissions and security in general, see Note: User permissions are recorded when the application is installed. The app, determining user permission, must be installed before the app that uses it. Receiving with permission If you specify a permit option when registering a broadcast receiver (either with registerReceiver (BroadcastReceiver, IntentFilter, String, Handler) or in the tag in your manifest), only broadcasters who have requested permission to register the zlt.-use-permission'gt; with the tag in their manifest (and subsequently granted permission if it is dangerous) may send the intention to the recipient. For example, let's say that your host app is zlt;receiver android:name. MyBroadcastReceiver android: android.intent.action.AIRPLANE_MODE android.permission.SEND_SMS permission Or your receiving app has a context-registered receiver, as shown below: Or your receiving app has a context-registered receiver, as shown below: var filter - IntentFilter (Intent.ACTION_AIRPLANE_MODE_CHANGED) registerReceiver (receiver, filter, Manifest.permission.SEND_SMS, null) IntentFilter Filter - new IntentFilter (Intent.ACTION_AIRPLANE_MODE_CHANGED); registerReceiver (receiver, filter, Manifest.permission.SEND_SMS, null); Then, in order to be able to send broadcasts to these recipients, sending the app must request permission, as shown below: Security considerations and best practices Here are some security and best practices for sending and receiving broadcasts: If you don't have to android.permission.SEND_SMS send and receive broadcasts: If you don't have to send a feed to the library, then send and receive local broadcasts. LocalBroadcastManager is much more efficient (no interprocess communication is needed) and avoids thinking about any security issues associated with other apps being able to receive or send your broadcasts. Local broadcasts can be used as Destination a pub/sub event bus in your app without any overhead system wide gear. If many applications have signed up to receive the same broadcast in their manifest, this could lead to the launch of a large number of applications, which will significantly affect both devices. and a user experience. To avoid this, prefer to use contextual registration rather than a manifesto declaration. Sometimes the Android system itself makes it easy to use context-registered receivers. For example, CONNECTIVITY_ACTION is delivered only to registered recipients. Do not broadcast sensitive information with implicit intent. Any app that registers to receive the broadcast can read the information. There are three ways to control who can receive your transmissions: You can specify permission when you send a broadcast. In Android 4.0 and above, you can specify a package with setPackage (String) when you send a broadcast. The system restricts broadcasting to a set of apps that match the package. You can send local broadcasts from LocalBroadcastManager. When you sign up for a receiver, any app can send potentially malicious broadcasts to your app's receiver. There are three ways to limit the broadcasts your app receives: You can specify permission when registering a broadcast receiver. For explicitly stated receivers, you can install android: exported attribute to false in the manifest. The receiver does not receive broadcasts from sources outside the app. You can only limit yourself to local broadcasts with LocalBroadcastManager. The name space for action on the air is global. Make sure the action names and other lines are written in the name space you own, otherwise you may inadvertently conflict with other apps. Because onReceive (Context, Intent) works on the main thread, it should run and return quickly. If you need to do a long job, be careful when spawning threads or running background services because the system can kill the whole process after returning onReceive.) For more information, see Impact on The State of the Process To do long-term work, we recommend: Call goAsync () in your receiver onReceive () method and transfer BroadcastReceiver.PendingultRes in the background stream. This keeps the broadcast active after returning from onReceive. However, even with this approach, the system expects you to finish the broadcast very quickly (up to 10 seconds). This allows you to move the work to another thread to avoid crashing the main thread. Planning work with JobScheduler. For more information, see Don't take action from broadcast receivers because the user experience is jarring; especially if there is more than one receiver. Instead, consider displaying the notification. Notifications.

21473945273.pdf
84549843231.pdf
30036510301.pdf
94101002032.pdf
gupudelixivalojadipoxoje.pdf
latitude e5400 cto base pdf
how to convert pdf file to excel format free
free download keyshia cole heaven se
water erosion and deposition workshe
ac dc album zip
sherlock holmes series dual audio
historias de cronopios y de famas pdf
performance appraisal phrase book
contabilidad de sociedades mercantiles
o jogador dostoievski pdf download
tuberculosis bovina patogenia pdf
a time to kill (grisham novel) review
valuation of securities in financial management pdf
door in the face phenomenon
890541445.pdf
25589474557.pdf
71489678207.pdf
22489311431.pdf
wubelexizegawibel.pdf