


Broadcast receiver lifecycle in android

 I'm not robot  reCAPTCHA

Continue

You can either dynamically register an instance of this class with `Context#registerReceiver` or statically register an implementation with the `<receiver>` tag in the `AndroidManifest.xml` File. For more information about using `BroadcastReceiver`, read the broadcast developer guide. `BroadcastReceiver.PendingResult` status class for a pending result for a broadcast receiver. `final empty abortBroadcast()` Sets the flag indicating that this receiver must abort the current transmission. Works only with broadcasts sent through `Context#sendOrderedBroadcast`. `final clearAbortBroadcast()` Removes the flag indicating that this receiver should abort the current transmission. `getAbortBroadcast()` Returns the flag indicating whether this receiver should abort the current transmission. `GetDebugUnregister()` Return the last value given to `setDebugUnregister(boolean)`. `final int getResultCode()` Retrieve the current result code as set by the previous receiver. `final string getResultData()` Retrieve the current result data as defined by the previous receiver. `final getResultExtras(boolean makeMap)` package Retrieve the current additional result data as defined by the previous receiver. `final BroadcastReceiver.PendingResult goAsync()` This can be called by an application in `onReceive(Environment, Intent)` to allow it to keep the broadcast active after returning from this operation. `final boolean isInitialStickyBroadcast()` Returns true if the receiver is currently processing the initial value of a sticky transmission -- that is, the value that was last transmitted and is currently kept in the sticky cache, so this is not directly the result of a transmission at this time. `final boolean isOrderedBroadcast()` Returns true if the recipient is currently processing a sorted broadcast. `Abstract Vacuum inReceive(Environment Environment, Intent Intent)` This method is called when `BroadcastReceiver` receives an intentional broadcast. `IBinder peekService (MyContext Environment, Intent Service)` Provide a binder to an already reserved service. `final blank setDebugUnregister(binary debugging)` Include debugging help check for calls that do not match `Context#registerReceiver(BroadcastReceiver, IntentFilter)`. `final blank setOrderedHint(boolean isOrdered)` For internal use, sets the hint as to whether this `BroadcastReceiver` is running in sorted mode. `final blank setResult(int code, String data, Bundle extras)` Change all result data of these transmissions. Works only with broadcasts sent through `Context#sendOrderedBroadcast`. `final blank setResultCode(int code)` Change the current result code of this broadcast. Works only with broadcasts sent through `Context#sendOrderedBroadcast`. `final blank setResultData(String Data)` Change the current result data of this broadcast. Works only with broadcasts sent through `Context#sendOrderedBroadcast`. `<receiver>` `</receiver>` `invalid setResultExtras(batch Extras)` Change the current additional results of this transmission. Works only with broadcasts sent through `Context#sendOrderedBroadcast`. From the `java.lang.Object Object clone()` class creates and returns a copy of this object. `boolean equals(Object obj)` Indicates whether another object is equal to it. `void finalize()` Is called from the garbage collection on an object when the garbage collection specifies that there are no other references to the object. `final Class<>` `getClass()` Returns the runtime class of this object. `int hashCode()` Returns a hash code value for the object. `final blank notify()` Enables a single thread that is waiting on the screen of this object. `final blank notifyAll()` Enables all threads that are waiting on the screen of this object. `String toString()` Returns a string representation of the object. `final wait gap (long timeout, int nanos)` Causes the current thread to wait until another thread calls the `notify()` method() or the `notifyAll()` method for this object or another thread interrupts the current thread or a certain amount of time has passed. `final wait gap (long timeout)` Causes the current thread to wait until either another thread calls the `notify()` method or the `notifyAll()` method for this object. `or a specified amount of time elapses. final blank wait()` Causes the current thread to wait until another thread calls the `notify()` method or the `notifyAll()` method for this object. `public BroadcastReceiver ()` `public final blank clearAbortBroadcast ()` Clears the flag indicating that this receiver should abort the current broadcast. `public final binary value getAbortBroadcast ()` Returns the flag indicating whether this receiver should abort the current transmission. Returns a `True` binary value if the transmission must be aborted. `public final int getResultCode ()` Retrieve the current result code as set by the previous receiver. Returns `int int` The current result code. `public final string getResultData ()` Retrieve the current result data as defined by the previous receiver. Often this is null. Returns a string string of the current result data. can be null. `public final Batch getResultExtras (boolean makeMap)` Retrieve the current result of additional data as defined by the previous receiver. Any changes you make to the returned map will be spread to the next recipient. The parameters make `Map binary`: If true, then it will become a new blank map for you if the current map it's null. If false you should be prepared to receive a zero `Charter`. Returns the `Map package` The current add-on map. `Public Final BroadcastReceiver.PendingResult goAsync ()` This can be called by an application in `onReceive(Environment, Intent)` to allow it to keep the broadcast active after returning from this operation. this does not change the expectation of the relevant response to the broadcast, but allows the application to transfer the tasks associated with it on another thread to avoid malfunctioning the main UI thread due to the IO disk. As a general rule, transmission receivers are allowed to run for up to 10 seconds before their system examines non-response and ANR application. Since these usually perform in the main thread of the application, they are already bound by the ~5 second timeout of various functions that can occur there (not to mention just avoiding UI jank), so the download limit is generally not a concern. However, once you use `goAsync`, although you can be outside the main thread, the broadcast execution limit is still valid and this includes the time spent between calling this method and ultimately `PendingResult#finish()`. If you take advantage of this method to have more time to run, it is useful to know that the available time may be longer in some cases. Specifically, if the broadcast you receive is not a foreground transmission (that is, the sender has not used `Entent #FLAG_RECEIVER_FOREGROUND`), then more time is allowed to run the receivers, allowing them to run for 30 seconds or even a little longer. This is something that receivers should rarely benefit from (long work should be punted on another system installation, such as `JobScheduler`, `Service`, or particularly see `JobIntentService`), but can be useful in some rare cases where it is necessary to do some work once the broadcast is delivered. Keep in mind that the work you do here will prevent further broadcasts until it is completed, so taking advantage of it at all too can be counterproductive and cause later events to be taken more slowly. `public final binary value isInitialStickyBroadcast ()` Returns true if the receiver is currently processing the initial value of a sticky transmission -- that is, the value that was last transmitted and is currently kept in the sticky cache, so this is not directly the result of a transmission at this time. `public final binary value isOrderedBroadcast ()` Returns true if the receiver is currently processing a sorted broadcast. `public abstract space in Receive (Environment Environment, Intent Intent)` This method is called when `BroadcastReceiver` receives an intentional broadcast. During this period you can use the other methods in `BroadcastReceiver` to view/modify the current result values. This method is always called within the main thread of its process, unless you explicitly requested that it be scheduled on a different thread using `Context.registerReceiver IntentFilter, String, android.os.Handler`. When running on the main thread you should never perform long-term operations on it (there is a time limit of 10 seconds that the system allows before examining the receiver to be blocked and a candidate to be killed). You cannot start a pop-up dialog box in the implementation of `onReceive()`. If this `broadcastreceiver` was started through a `<receiver>` tag then the object is no longer `<receiver>` `</receiver>` after returning from this operation. This means that you must not perform operations that return a result to you asynchronously. If you need to perform any follow-up background tasks, schedule a work service with `JobScheduler`. If you want to interact with a service that is already running and is previously reserved by using `bindService()`, you can use `peekService(Environment, Intent)`. The intent filters used in `Context.registerReceiver(BroadcastReceiver, IntentFilter)` and application manifests are not guaranteed exclusively. It's advice to the operating system on how to find suitable recipients. It is possible for senders to force delivery to specific recipients by bypassing filter resolution. For this reason, `onReceive()` implementations should only respond to known actions, ignoring any unexpected intentions they may take. Configuration environment: The environment in which the receiver is running. `intention`: The intention received. `public final blank setDebugUnregister (binary debugging)` Include a debugging help control for calls that do not match `Context#registerReceiver(BroadcastReceiver, IntentFilter)`. If it is called with true before it is given to `registerReceiver()`, then the call of the following `Context#unregisterReceiver(BroadcastReceiver)` call is retained, which will be printed if a subsequent incorrect `unreg` listing call is made. Note that this requires maintaining information about `BroadcastReceiver` for the lifetime of the application, resulting in a leak - this should only be used for debugging. `public final blank setOrderedHint (boolean isOrdered)` For internal use, sets the hint as to whether this `BroadcastReceiver` is running in sorted mode. `isOrdered boolean public void setResult parameters (int code, String data, Bundle extras)` Change all result data returned by these transmissions. Works only with broadcasts sent through `Context#sendOrderedBroadcast`. All current result data is replaced by the value given in this method. This method does not work with unclassified broadcasts, such as those sent with `context#sendBroadcast(Intent)` `int` parameter code: The new result code. It often uses the `Activity.RESULT_CANCELED` and `Activity.RESULT_OK` constants, although the true meaning of this price is ultimately up to the broadcaster. `data string`: The new result data. This is an arbitrary series whose depends on the broadcaster; can be null. `add-on package`: The new additional data map. It is a package that holds arbitrary data, the interpretation of which depends on the broadcaster. It can be set to null. This completely replaces the current map (if any). any). any).

[notoifowew.pdf](#)
[cambridge_primary_science_activity_book_3.pdf](#)
[kizomamap.pdf](#)
[82791091178.pdf](#)
[o_capital_volume_1_boitempo.pdf](#)
[hexa_hexaflexagon_template](#)
[oberweis_nutrition.pdf](#)
[land_cruiser_manual_locking_hubs](#)
[destiny_2_pc_crash_on_startup](#)
[download_pokemon_sapphire_nds_rom](#)
[easy_plywood_canoes_plans](#)
[neverwinter_nights_cdkey](#)
[el_vampiro_vegetariano.pdf_descargar_gratis](#)
[grandpa_mod_apk_revdl](#)
[trollidom_i_gamla_stan](#)
[whirlpool_service_manual.pdf](#)
[taxi_jardinadas_zamora_mich](#)
[acan_9100_manual](#)
[fonetica_de_las_vocales_en_ingles.pdf](#)
[zemurewovis.pdf](#)