Android Developers

Background Execution Limits

Whenever an app runs in the background, it consumes some of the device's limited resources, like RAM. This can result in an impaired user experience, especially if the user is using a resource-intensive app, such as playing a game or watching video. To improve the user experience, Android 8.0 (API level 26) imposes limitations on what apps can do while running in the background. This document describes the changes to the operating system, and how you can update your app to work well under the new limitations.

In this document

Overview

Many Android apps and services can be running simultaneously. For example, a user could be playing a game in one window while browsing the web in another window, and using a third app to play music. The more apps are running at once, the more load is placed on the system. If additional apps or services are running in the background, this places additional loads on the system, which could result in a poor user experience; for example, the music app might be suddenly shut down.

To lower the chance of these problems, Android 8.0 places limitations on what apps can do while users aren't directly interacting with them. Apps are restricted in two ways:

- Background Service Limitations (#services): While an app is idle, there are limits to its use of background services. This does not apply to foreground services, which are more noticeable to the user.
- Broadcast Limitations (#broadcasts): With limited exceptions, apps cannot use their manifest to register for implicit broadcasts. They can still register for these broadcasts at runtime, and they can use the manifest to register for explicit broadcasts targeted specifically at their app.

> **Note:** By default, these restrictions only apply to apps that target Android 8.0 (API level 26) or higher. However, users can enable most of these restrictions for any app from the **Settings** screen, even if the app targets an API level lower than 26.

In most cases, apps can work around these limitations by using `JobScheduler` `(https://developer.android.com/reference/android/app/job/JobScheduler.html)` jobs. This approach lets an app arrange to perform work when the app isn't actively running, but still gives the system the leeway to schedule these jobs in a way that doesn't affect the user experience. Android 8.0 offers several improvements to `JobScheduler` `(https://developer.android.com/reference/android/app/job/JobScheduler.html)` that make it easier to replace services and broadcast receivers with scheduled jobs; for more information, see JobScheduler improvements (https://developer.android.com/about/versions/o/android-8.0.html#jobscheduler).

Background Service Limitations

Services running in the background can consume device resources, potentially resulting in a worse user experience. To mitigate this problem, the system applies a number of limitations on services.

The system distinguishes between *foreground* and *background* apps. (The definition of background for purposes of service limitations is distinct from the definition used by memory management (https://developer.android.com/topic/performance/memory-overview.html); an app might be in the background as pertains to memory management, but in the foreground as pertains to its ability to launch services.) An app is considered to be in the foreground if any of the following is true:

- It has a visible activity, whether the activity is started or paused.
- It has a foreground service.
- Another foreground app is connected to the app, either by binding to one of its services or by making use of one of its content providers. For example, the app is in the foreground if another app binds to its:

- IME (https://developer.android.com/guide/topics/text/creating-input-method.html)
- Wallpaper service
- Notification listener
- Voice or text service

If none of those conditions is true, the app is considered to be in the background.

While an app is in the foreground, it can create and run both foreground and background services freely. When an app goes into the background, it has a window of several minutes in which it is still allowed to create and use services. At the end of that window, the app is considered to be *idle*. At this time, the system stops the app's background services, just as if the app had called the services' `Service.stopSelf()` (https://developer.android.com/reference/android/app/Service.html#stopSelf()) methods.

Bound services are not affected

These rules do not affect bound services in any way. If your app defines a bound service, other components can bind to that service whether or not your app is in the foreground.

Under certain circumstances, a background app is placed on a temporary whitelist for several minutes. While an app is on the whitelist, it can launch services without limitation, and its background services are permitted to run. An app is placed on the whitelist when it handles a task that's visible to the user, such as:

- Handling a high-priority Firebase Cloud Messaging (FCM) (https://firebase.google.com/docs/cloud-messaging/) message.
- Receiving a broadcast, such as an SMS/MMS message.
- Executing a `PendingIntent` (https://developer.android.com/reference/android/app/PendingIntent.html) from a notification.
- Starting a `VpnService` (https://developer.android.com/reference/android/net/VpnService.html) before the VPN app promotes itself to the foreground.

In many cases, your app can replace background services with `JobScheduler` (https://developer.android.com/reference/android/app/job/JobScheduler.html) jobs. For example, CoolPhotoApp needs to check whether the user has received shared photos from friends, even if the app isn't running in the foreground. Previously, the app used a background service which checked with the app's cloud storage. To migrate to Android 8.0 (API level 26), the developer replaces the background service with a scheduled job, which is launched periodically, queries the server, then quits.

Prior to Android 8.0, the usual way to create a foreground service was to create a background service, then promote that service to the foreground. With Android 8.0, there is a complication; the system doesn't allow a background app to create a background service. For this reason, Android 8.0 introduces the new method `startForegroundService()` (https://developer.android.com/reference/android/content/Context.html#startForegroundService(android.content.Intent)) to start a new service in the foreground. After the system has created the service, the app has five seconds to call the service's `startForeground()` (https://developer.android.com/reference/android/app/Service.html#startForeground(int, android.app.Notification)) method to show the new service's user-visible notification. If the app does *not* call `startForeground()` (https://developer.android.com/reference/android/app/Service.html#startForeground(int, android.app.Notification)) within the time limit, the system stops the service and declares the app to be ANR (https://developer.android.com/training/articles/perf-anr.html).

IntentService is subject to these restrictions

`IntentService` (https://developer.android.com/reference/android/app/IntentService.html) is a service, and is therefore subject to the new restrictions on background services. As a result, many apps that rely on `IntentService` (https://developer.android.com/reference/android/app/IntentService.html) do not work properly when targeting Android 8.0 or higher. For this reason, Android Support Library 26.0.0 (https://developer.android.com/topic/libraries/support-library/revisions.html#26-0-0) introduces a new `JobIntentService` (https://developer.android.com/reference/android/support/v4/app/JobIntentService.html) class, which provides the same functionality as `IntentService` (https://developer.android.com/reference/android/app/IntentService.html) but uses jobs instead of services when running on Android 8.0 or higher.

Broadcast Limitations

If an app registers to receive broadcasts, the app's receiver consumes resources every time the broadcast is sent. This can cause problems if too many apps register to receive broadcasts based on system events; a system event that triggers a broadcast can cause all of those apps to consume resources in rapid succession, impairing the user experience. To mitigate this problem, Android 7.0 (API level 25) placed limitations on broadcasts, as described in Background Optimization (https://developer.android.com/topic/performance/background-optimization.html). Android 8.0 (API level 26) makes these limitations more stringent.

- Apps that target Android 8.0 or higher can no longer register broadcast receivers for implicit broadcasts in their manifest. An *implicit broadcast* is a broadcast that does not target that app specifically. For example, `ACTION_PACKAGE_REPLACED` `(https://developer.android.com/reference/android/content/Intent.html#ACTION_PACKAGE_REPLACED)` is an implicit broadcast, since it is sent to all registered listeners, letting them know that some package on the device was replaced. However, `ACTION_MY_PACKAGE_REPLACED` `(https://developer.android.com/reference/android/content/Intent.html#ACTION_MY_PACKAGE_REPLACED)` is not an implicit broadcast, since it is sent only to the app whose package was replaced, no matter how many other apps have registered listeners for that broadcast.
- Apps can continue to register for explicit broadcasts in their manifests.
- Apps can use `Context.registerReceiver()` `(https://developer.android.com/reference/android/content/Context.html#registerReceiver(android.content.BroadcastReceiver, android.content.IntentFilter))` at runtime to register a receiver for any broadcast, whether implicit or explicit.
- Broadcasts that require a signature permission (https://developer.android.com/guide/topics/manifest/permission-element.html#plevel) are exempted from this restriction, since these broadcasts are only sent to apps that are signed with the same certificate, not to all the apps on the device.

In many cases, apps that previously registered for an implicit broadcast can get similar functionality by using a `JobScheduler` `(https://developer.android.com/reference/android/app/job/JobScheduler.html)` job. For example, a social photo app might need to perform cleanup on its data from time to time, and prefer to do this when the device is connected to a charger. Previously, the app registered a receiver for `ACTION_POWER_CONNECTED` `(https://developer.android.com/reference/android/content/Intent.html#ACTION_POWER_CONNECTED)` in its manifest; when the app received that broadcast, it would check whether cleanup was necessary. To migrate to Android 8.0 or higher, the app removes that receiver from its manifest. Instead, the app schedules a cleanup job that runs when the device is idle and charging.

> **Note:** A number of implicit broadcasts are currently exempted from this limitation. Apps can continue to register receivers for these broadcasts in their manifests, no matter what API level the apps are targeting. For a list of the exempted broadcasts, see Implicit Broadcast Exceptions (https://developer.android.com/guide/components/broadcast-exceptions.html).

Migration Guide

By default, these changes only affect apps that target Android 8.0 (API level 26) or higher. However, users can enable these restrictions for any app from the **Settings** screen, even if the app targets an API level lower than 26. You may need to update your app to comply with the new limitations.

Check to see how your app uses services. If your app relies on services that run in the background while your app is idle, you will need to replace them. Possible solutions include:

- If your app needs to create a foreground service while the app is in the background, use the new `NotificationManager.startServiceInForeground()` method instead of creating a background service and trying to promote it to the foreground.
- If the service is noticeable by the user, make it a foreground service. For example, a service that plays audio should always be a foreground service. Create the service with `NotificationManager.startServiceInForeground()` instead of `startService()` `(https://developer.android.com/reference/android/content/Context.html#startService(android.content.Intent))`.
- Find a way to duplicate the service's functionality with a scheduled job. If the service is not doing something immediately noticeable to the user, you should generally be able to use a scheduled job instead.
- Use FCM (https://firebase.google.com/docs/cloud-messaging/) to selectively wake your application up when network events occur, rather than polling in the background.

- Defer background work until the application is naturally in the foreground.

Review the broadcast receivers defined in your app's manifest. If your manifest declares a receiver for an implicit broadcast, you must replace it. Possible solutions include:

- Create the receiver at runtime by calling `Context.registerReceiver()` (https://developer.android.com/reference/android/content/Context.html#registerReceiver(android.content.BroadcastReceiver, android.content.IntentFilter)), instead of declaring the receiver in the manifest.
- Use a scheduled job to check for the condition that would have triggered the implicit broadcast.

Follow @AndroidDev on Twitter

Follow Android Developers on Google+

Check out Android Developers on YouTube