Differences between Intent and PendingIntent

Asked 9 years, 4 months ago Modified 10 months ago Viewed 65k times

Part of Mobile Development Collective

115

I read through some articles. All seem to do the same thing and I was wondering what is the difference between starting the service as below:

```
Intent intent = new Intent(this, HelloService.class);
startService(intent);
```

or below:

```
Calendar cal = Calendar.getInstance();
Intent intent = new Intent(this, MyService.class);
PendingIntent pintent = PendingIntent.getService(this, 0, intent, 0);
AlarmManager alarm = (AlarmManager)getSystemService(Context.ALARM_SERVICE);
alarm.setRepeating(AlarmManager.RTC_WAKEUP, cal.getTimeInMillis(), 30*1000, pintent);
```

As I read through, these two do the same thing, if in the service you return a parameter START_STICKY;

MD android android-service

Share Edit Follow

edited Mar 30, 2021 at 20:05
Priyanka
1,789 ● 1 ● 7 ● 12

asked Jun 17, 2014 at 6:42
user3629316
1,245 ● 2 ● 10 ● 8

There's no difference. What makes you think there would be? In the first case you are starting it 'now' and in the second you're just scheduling it for a later time/data. – Squonk Jun 17, 2014 at 7:00

Possible duplicate of What is an Android PendingIntent? – James Moore Jul 7, 2017 at 17:34

6 Answers

Sorted by: Highest score (default)

190

Intent

An Android Intent is an object carrying an intent, i.e. a message from one component to another component either inside or outside of the application. Intents can communicate messages among any of the three core components of an application -- Activities, Services, and BroadcastReceivers.

The intent itself, an Intent object, is a passive data structure. It holds an abstract description of an operation to be performed.

For example: say you have an Activity that needs to launch an email client and send an email. To do this, your Activity would send an Intent with the action `ACTION_SEND`, along with the appropriate chooser, to the Android Intent Resolver:

```
Intent intent = new Intent(Intent.ACTION_SENDTO);
intent.setData(Uri.parse("mailto:")); // only email apps should handle this
```

The specified chooser gives the proper interface for the user to pick how to send your email data.

EXPLICIT INTENTS

```
// Explicit Intent by specifying its class name
   Intent i = new Intent(this, TargetActivity.class);
   i.putExtra("Key1", "ABC");
   i.putExtra("Key2", "123");

// Starts TargetActivity
   startActivity(i);
```

IMPLICIT INTENTS

```
// Implicit Intent by specifying a URI
   Intent i = new Intent(Intent.ACTION_VIEW,
   Uri.parse("http://www.example.com"));

// Starts Implicit Activity
   startActivity(i);
```

Pending Intent

A PendingIntent is a token that you give to a *foreign* application (e.g. NotificationManager, AlarmManager, Home Screen AppWidgetManager, or other 3rd party applications), which allows the foreign application to use your application's permissions to execute a predefined piece of code.

> By giving a PendingIntent to another application, you are granting it the right to perform the operation you have specified as if the other application was yourself (with the same permissions and identity). As such, you should be careful about how you build the PendingIntent: almost always, for example, the base Intent you supply should have the component name explicitly set to one of your own components, to ensure it is ultimately sent there and nowhere else.

Example for Pending Intent : http://android-pending-intent.blogspot.in/

Source : Android Intents and Android Pending Intents

Share Edit Follow edited Dec 23, 2022 at 8:57 answered Jun 17, 2014 at 10:53


starball
22.8k ● 9 ● 49 ● 309


Siddharth_Vyas
9,982 ● 10 ● 39 ● 69

32

`PendingIntent` is a wrapper of `Intent`. The foreign app that receives the `PendingIntent`, doesn't know the content of `Intent` which is wrapped by `PendingIntent`. The mission of foreign app is to send back the intent to owner when some conditions are met (For example: alarm with schedule, or notification with click...). The conditions are given by owner but processed by foreign app (For example: alarm, notification).

If foreign app sent intent to your app, mean that foreign app know about the content of the intent. and foreign app make decision to send intent then your app must process intent to meet some conditions => your app get performance resource of system.

Share Edit Follow

edited Aug 21, 2016 at 17:23


Yogesh Umesh Vaity
42k ● 21 ● 145 ● 106

answered Jan 25, 2016 at 14:01


HungNM2
3,181 ● 1 ● 32 ● 21

12

Another simple difference:

- Normal intent will die as soon as the app is killed.
- Pending intents never die. They will be alive as long as it's needed by alarm service, location service, or any other services.

Share Edit Follow

edited Dec 7, 2018 at 1:29


Pang
9,627 ● 146 ● 81 ● 122

answered Oct 30, 2017 at 2:50


Zumry Mohamed
9,338 ● 5 ● 46 ● 51

6

There's another major difference between `Intent` and `PendingIntent` which is better to be aware of, otherwise your app's design might become **vulnerable**. The problem is well described in Android Nesting Intents article.

Note that `PendingIntent.send()` method doesn't accept a `Context` instance, it uses a context provided during the intent creation instead. It allows a third party component to execute an action associated with a pending intent within the intent creator's context.

Let's imagine a third party service which performs some job and then starts an activity specified by your app as an intent. If the *callback* activity is provided as a basic `Intent`, the service can only start it using its own context and such a design has two drawbacks:

- It forces the *callback* activity to be defined as `exported` so it can be started using a third party context. As a result the activity can be started not only by the service it was intended for, but by any other app installed on the device.
- Any activity defined by the third party service app can be used as a *callback* activity, even non-exported one, since it's started using the third party context.

Both problems can be easily solved by specifying the *callback* activity as `PendingIntent` instead.

Share Edit Follow

answered Mar 1, 2021 at 14:39

Valeriy Katkov
34.9k ● 21 ● 103 ● 123

I think the `exported` character is the most significant – twlkyao Jun 22, 2021 at 6:31

This is the best answer in my opinion, and the referred article is priceless. – HZhang Aug 4, 2021 at 18:05

2

Functionally, there is no difference.

The meaning of PendingIntent is that, you can handle it to other application that later can use it as if the other application was yourself. Here is the relevant explanation from the documentation:

> By giving a PendingIntent to another application, you are granting it the right to perform the operation you have specified as if the other application was yourself (with the same permissions and identity). As such, you should be careful about how you build the PendingIntent: almost always, for example, the base Intent you supply should have the component name explicitly set to one of your own components, to ensure it is ultimately sent there and nowhere else.
>
> A PendingIntent itself is simply a reference to a token maintained by the system describing the original data used to retrieve it.

So PendingIntent is just a reference to the data that represents the original Intent (that used to create the PendingIntent).

Share Edit Follow

answered Jun 17, 2014 at 10:25

kupsef
3,367 ● 1 ● 22 ● 31

6 Saying there is functionally no difference is incorrect. If functions of both are same then why have two in 1st place? Most important difference in that PendingIntent is executed by remote component (Like NotificationManager) with the same permissions as that of the component that hands it over (The one that creates the notification). – Aniket Thakur Apr 5, 2015 at 5:17

1

Starting services regularly via AlarmManager

As with activities the Android system may terminate the process of a service at any time to save resources. For this reason you cannot simply use a `TimerTask` in the service to ensure that it is executed on a regular basis.

So, for correct scheduling of the Service use the `AlarmManager` class.

UPDATE:

So there is no actual difference between the two. But depending on whether you want to ensure the execution of the service or not, you can decide what to use as for the *former* there is no guarantee and for the *later* it is.

More info at AndroidServices.

Share Edit Follow

edited Jun 17, 2014 at 7:03

answered Jun 17, 2014 at 6:52

sjain

23.2k ● 28 ● 108 ● 187

2 This doesn't actually answer the OP's question which is "what is the difference" between directly starting a service and starting a service with an alarm. Also the OP has probably seen the article you link to as the code in the article is almost identical to what the OP has posted. – Squonk Jun 17, 2014 at 6:58

Do you mean starting a service from AlarmManager is safer, and less likely to be killed than from an activity? I think it is wrong. Can you please explain. @VedPrakash. More over, I think the context that you pass while creating an intent to start the service is more important. Using the context of the application(getApplicationContext()) rather than that of an activity(this), should be safer. – Parth Kapoor Jun 17, 2014 at 7:16

@Eu.Dr. I recommend you to use alarm manager that will be trigger at every time X... execute the task.. Why? because if you use services it might be closed at some point and you might end up skipping some updates at a certain time (unknown). For context doubt, never use `getApplicationContext()` or use it when you want it strictly, just read - when-to-call-activity-context-or-application-context (stackoverflow.com/questions/7298731/...). – sjain Jun 17, 2014 at 7:50