Введение в разработку под Android

Семинар 9.

Макаров И.О. 13.11.2020

На этом семинаре

- Летучка(большая)
- Сервисы
- Background, Foreground, Bounded сервисы
- LifeCycle сервисов
- Примеры

Летучка #1

- Внезапно нам понадобилось разместить аватарку пользователя так, чтобы отступ от левого края экрана был в два раза больше, чем от правого, а сама картинка была по высоте в два раза больше, чем по ширине. Какие атрибуты ConstraintLayout нужно использовать?
- Чем отличаются ориентиры и барьеры в ConstraintLayout?
- Какие методы необходимо реализовать при наследовании RecyclerView.Adapter?
- К нам пришел заказчик и хочет "переключатель", "выпадашку" и "ползунок". Какие элементы интерфейсов нужно использовать?
- В нашем приложении есть экран настроек. Какой класс предоставляет доступ к конкретным значениям?

Летучка #2

- Напишите, с помощью чего можно отправить задание на исполнение main-потоку.
- Чтобы загрузить файл с сервера необходимо сэмулировать отправку POST-запроса из HTML-формы. Как сделать это с помощью DownloadManager?
- Приложению необходимо раз в день осуществлять синхронизацию с сервером. Мы хотим, чтобы это происходило, когда телефон подключен к Wi-Fi сети и у устройства имеется высокий заряд батарейки. У нас имеется код синхронизации, какие дальнейшие шаги необходимо совершить?
- Чтобы запустить activity нужно вызвать startActivity. Что нужно вызвать, чтобы запустить сервис?
- Существует ли механизм прерывания исполнения задачи в цепочке задач(ассоциированной с цепочкой future объектов, которые можно, например, получить, поставив задачу с помощью WorkManager). Если да, то коротко опишите, как оно может быть устроено.

Services Overview

https://developer.android.com/guide/components/services

A Service is an application component that can perform long-running operations in the background. It does not provide a user interface.

Once started, a service might continue running for some time, even after the user switches to another application.

Additionally, a component can bind to a service to interact with it and even perform interprocess communication (IPC).

For example, a service can handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background.

Caution: A service runs in the main thread of its hosting process; the service does **not** create its own thread and does **not** run in a separate process unless you specify otherwise. You should run any blocking operations on a separate thread within the service to avoid Application Not Responding (ANR) errors.

Types of Services

https://developer.android.com/guide/components/services

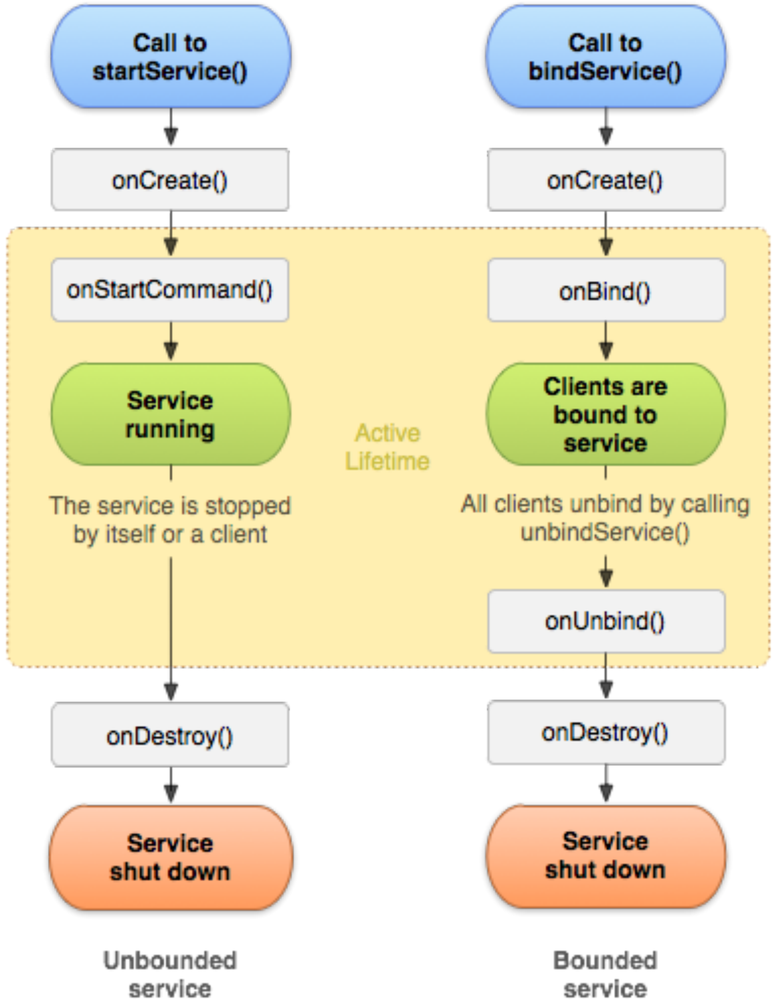
- Foreground
- Background
- Bounded

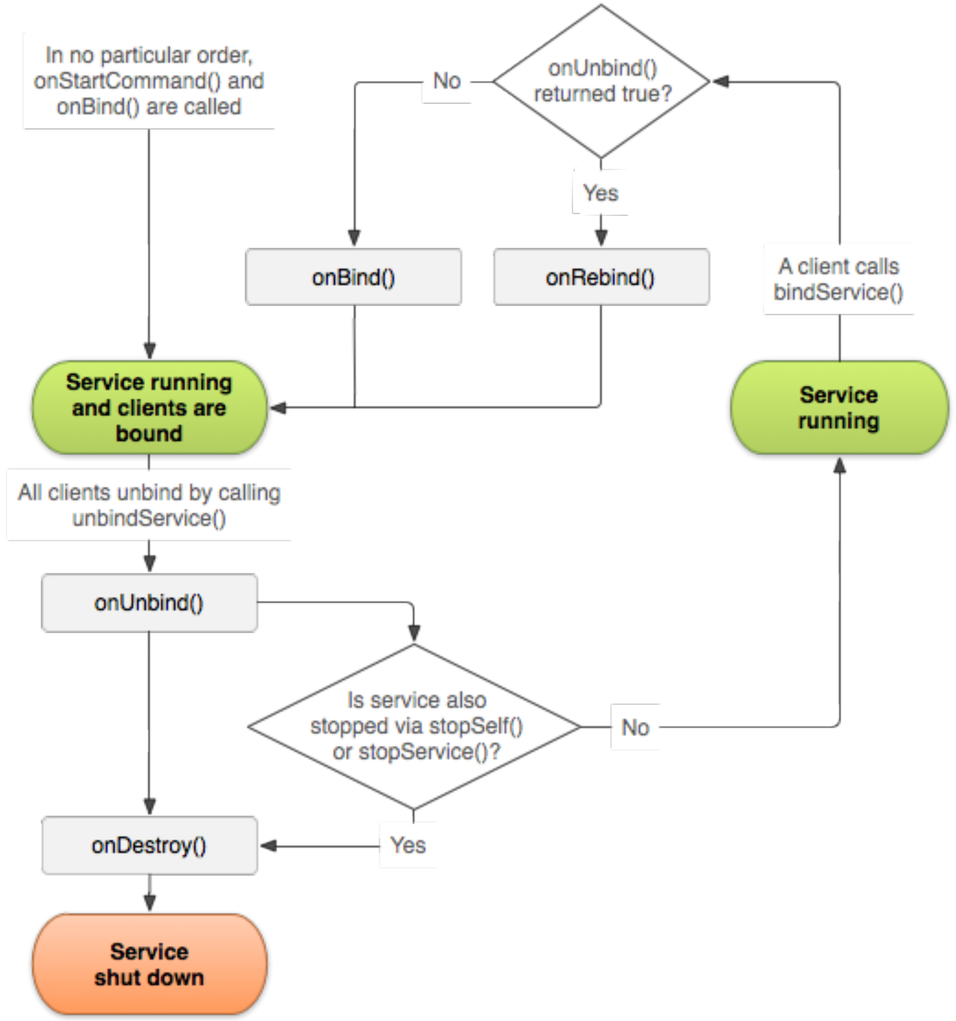
Note: The WorkManager API offers a flexible way of scheduling tasks, and is able to run these jobs as foreground services if needed. In many cases, using WorkManager is preferable to using foreground services directly.

Note: If your app targets API level 26 or higher, the system imposes restrictions on running background services when the app itself isn't in the foreground. In most situations, for example, you shouldn't access location information from the background. Instead, schedule tasks using WorkManager.

Managing the lifecycle of a service

https://developer.android.com/guide/components/services





Declaring a service in the manifest

https://developer.android.com/guide/components/services

```
<manifest ... >
  ...
  <application ... >
      <service android:name=".ExampleService" />
      ...
  </application>
</manifest>
```

Note: Users can see what services are running on their device. If they see a service that they don't recognize or trust, they can stop the service. In order to avoid having your service stopped accidentally by users, you need to add the **android:description** attribute to the **<service>** element in your app manifest. In the description, provide a short sentence explaining what the service does and what benefits it provides.

Caution: To ensure that your app is secure, always use an explicit intent when starting a **Service** and don't declare intent filters for your services. Using an implicit intent to start a service is a security hazard because you cannot be certain of the service that responds to the intent, and the user cannot see which service starts. Beginning with Android 5.0 (API level 21), the system throws an exception if you call **bindService()** with an implicit intent.

Notice about onStartCommand(…)

https://developer.android.com/guide/components/services

onStartCommand() method must return an integer. The integer is a value that describes how the system should continue the service in the event that the system kills it. The return value from onStartCommand() must be one of the following constants:

START_NOT_STICKY
If the system kills the service after onStartCommand() returns, *do not* recreate the service unless there are pending intents to deliver. This is the safest option to avoid running your service when not necessary and when your application can simply restart any unfinished jobs.

START_STICKY
If the system kills the service after onStartCommand() returns, recreate the service and call onStartCommand(), but *do not* redeliver the last intent. Instead, the system calls onStartCommand() with a null intent unless there are pending intents to start the service. In that case, those intents are delivered. This is suitable for media players (or similar services) that are not executing commands but are running indefinitely and waiting for a job.

START_REDELIVER_INTENT
If the system kills the service after onStartCommand() returns, recreate the service and call onStartCommand() with the last intent that was delivered to the service. Any pending intents are delivered in turn. This is suitable for services that are actively performing a job that should be immediately resumed, such as downloading a file.

Starting a service

https://developer.android.com/guide/components/services

You can start a service from an activity or other application component by passing an `Intent` to `startService()` or `startForegroundService()`. The Android system calls the service's `onStartCommand()` method and passes it the `Intent`, which specifies which service to start.

```
...
Intent intent = new Intent(…);
startService(intent);
...
```

Note: If your app targets API level 26 or higher, the system imposes restrictions on using or creating background services unless the app itself is in the foreground. If an app needs to create a foreground service, the app should call **startForegroundService()**. That method creates a background service, but the method signals to the system that the service will promote itself to the foreground. Once the service has been created, the service must call its **startForeground()** method within five seconds.

Stopping a service

https://developer.android.com/guide/components/services

A started service must manage its own lifecycle. That is, the system doesn't stop or destroy the service unless it must recover system memory and the service continues to run after onStartCommand() returns. The service must stop itself by calling stopSelf(), or another component can stop it by calling stopService().

Once requested to stop with stopSelf() or stopService(), the system destroys the service as soon as possible.

If your service handles multiple requests to onStartCommand() concurrently, you shouldn't stop the service when you're done processing a start request, as you might have received a new start request (stopping at the end of the first request would terminate the second one). To avoid this problem, you can use stopSelf(int) to ensure that your request to stop the service is always based on the most recent start request. That is, when you call stopSelf(int), you pass the ID of the start request (the startId delivered to onStartCommand()) to which your stop request corresponds. Then, if the service receives a new start request before you are able to call stopSelf(int), the ID doesn't match and the service doesn't stop.

Caution: To avoid wasting system resources and consuming battery power, ensure that your application stops its services when it's done working. If necessary, other components can stop the service by calling **stopService()**. Even if you enable binding for the service, you must always stop the service yourself if it ever receives a call to **onStartCommand()**.

Дополнительно можно

https://developer.android.com/courses/fundamentals-training

- Прочитать про JobIntentService, Messenger и др.
- AIDL
- Расширения Binder