Service

Что такое Service

Сервис (Service) - один из 4 главных компонентов андроид приложения, а это значит что он обязательно регистрируется в манифесте.
Сервис – это какая-то задача, которая работает в фоне и не использует UI. Запускать и останавливать сервис можно из приложений и других сервисов. Также можно подключиться к уже работающему сервису и взаимодействовать с ним.

```
<service
        android:exported=["true" | "false"]
        android:icon="drawable resource"
        android:label="string resource"
        android:name="string"
        android:process="string" >
<intent-filter>
<meta-data>

</service>
```

`android:exported` - показывает могут ли другие приложения запускать сервис или взаимодействовать с ним.

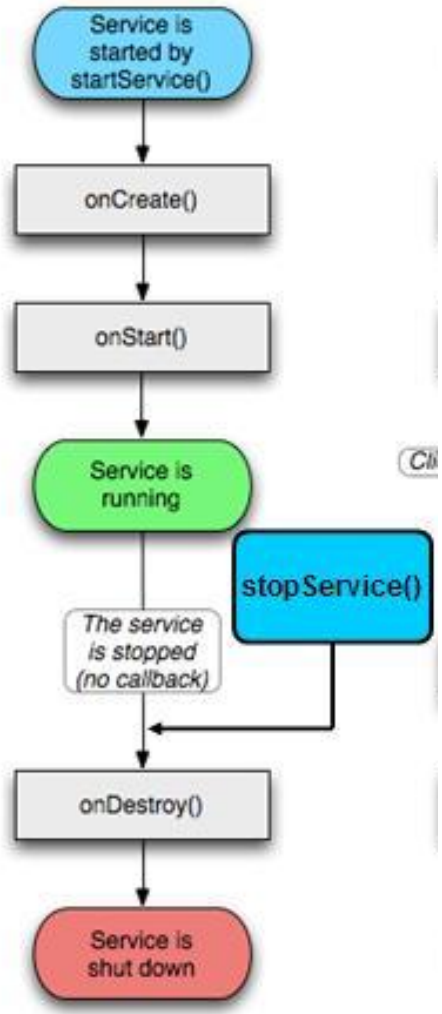
`android:process` - можно указать что сервис будет запущен в другом процессе, отличном от процесса самого приложения. (android:process=":newproc")

Назначение

Позволяет выполнять длительные операции, которые не требуют взаимодействия с UI.
Другой компонент приложения может запустить сервис и он будет работать даже если приложение будет свёрнуто.
Также компонент (обычно активити) может подключится к сервису (bind) и взаимодействовать с ним. Также возможно взаимодействие между сервисом и компонентом из другого приложения (но можно сделать сервис приватным).
Например, сервис может производить передачу данных по сети, проигрывать музыку из бекграунда (не взаимодействуя с ui).

Сервис работает в UI потоке, так что для длительных операций в нем всё равно необходимо создавать новый поток.

Сервисы бывают двух типов.

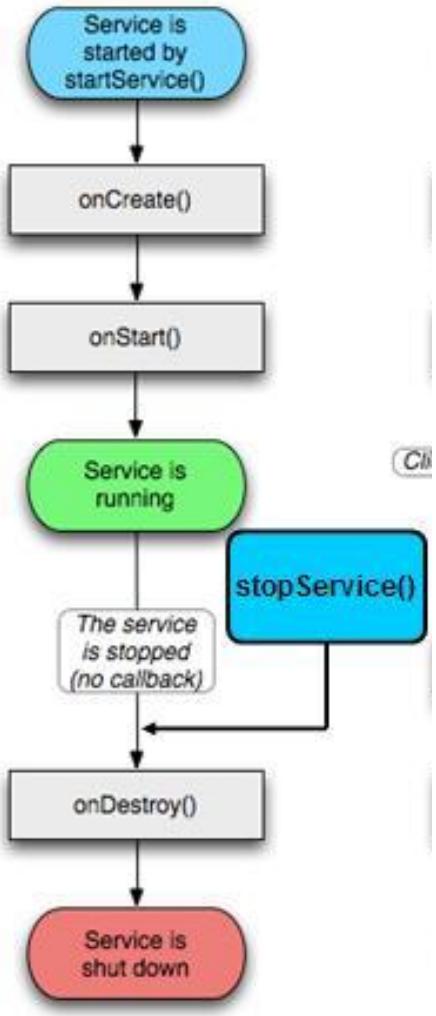


Первый тип: *ЗАПУСК*

Какой-то компонент ***запускает*** сервис с помощью startService().

Как только он запущен, сервис может работать очень долго, даже когда компонент, который его запустил завершился (или был уничтожен системой).

Обычно такие сервисы выполняют одну операцию (передача файла по сети, например) и после выполнения самостоятельно останавливаются с помощью stopSelf() или stopSelfResult(). При завершении будет вызван метод onDestroy()



Жизненный цикл сервиса первого типа (*Запуск*)

Как только был вызван метод startService() (в этот метод передается интент, так что можно положить необходимые данные) при первом создании будет вызван метод onCreate() а затем метод onStartCommand(Intent, int, int) в который передается интент, флаг (0, START_FLAG_REDELIVERY, or START_FLAG_RETRY) и startId (идентификатор для запуска сервиса, используется с stopSelfResult(int)) . Вернуть необходимо константу, которая описывает что необходимо сделать, если сервис вдруг уничтожится (START_STICKY (используется для сервисов, которые должны быть вызваны и остановлены только методами); START_NOT_STICKY, START_REDELIVER_INTENT (используются для сервисов, которые должны быть запущены пока обрабатывают присланные данные)).
С этого момента сервис будет работать до тех пор пока не будет остановлен одним из этих методов stopService() или stopSelf() . Неважно сколько раз сервис был вызван, он будет завершен при вызове этих методов. Есть метод stopSelf(int), который позволяет останавливать сервис по startId.

Метод onStartCommand(Intent intent, int flags, int startId)

Система может убить сервис, если ей будет не хватать памяти. Но мы можем сделать так, чтобы сервис ожил, когда проблема с памятью будет устранена. И более того, не просто ожил, а еще и снова начал выполнять незавершенные вызовы **startService**.

Мы вызываем startService, срабатывает **onStartCommand** и возвращает одну из следующих констант:

START_NOT_STICKY – сервис не будет перезапущен после того, как был убит системой

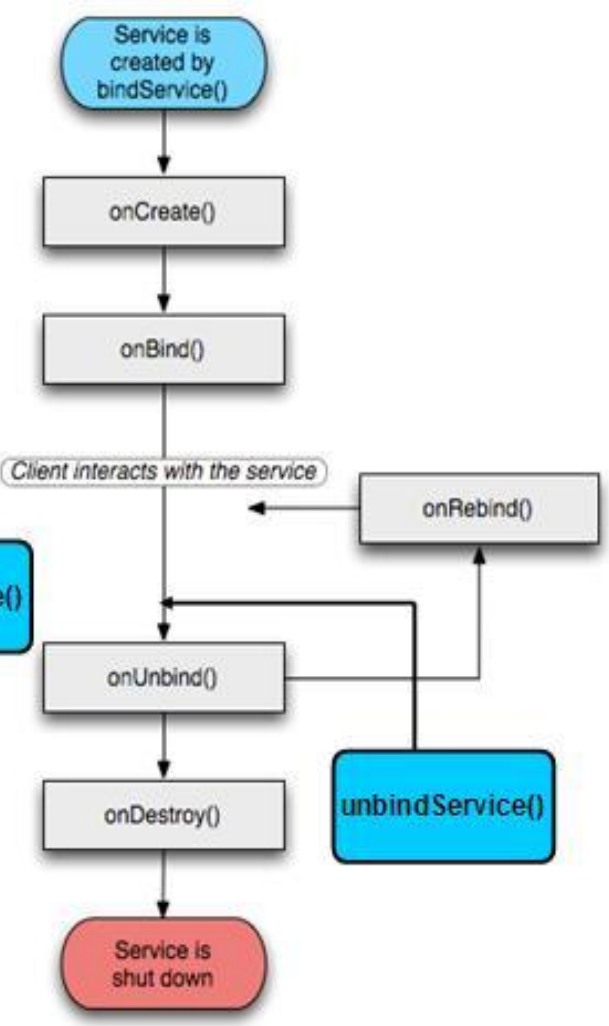
START_STICKY – сервис будет перезапущен после того, как был убит системой

START_REDELIVER_INTENT – сервис будет перезапущен после того, как был убит системой. Кроме этого, сервис снова получит все вызовы startService, которые не были завершены методом **stopSelf(startId)**.

Второй параметр flags дает нам понять, что это повторная попытка вызова onStartCommand. flags может принимать значения 0, START_FLAG_RETRY или START_FLAG_REDELIVERY.

Если START_FLAG_REDELIVERY пришел в методе onStartCommand, значит, прошлый вызов этого метода вернул START_REDELIVER_INTENT, но не был завершен успешно методом stopSelf(startId). Если во время убийства сервиса будет висеть несколько таких работающих START_FLAG_REDELIVERY вызовов, для которых еще не был выполнен stopSelf, то все они будут снова отправлены в сервис при его восстановлении.

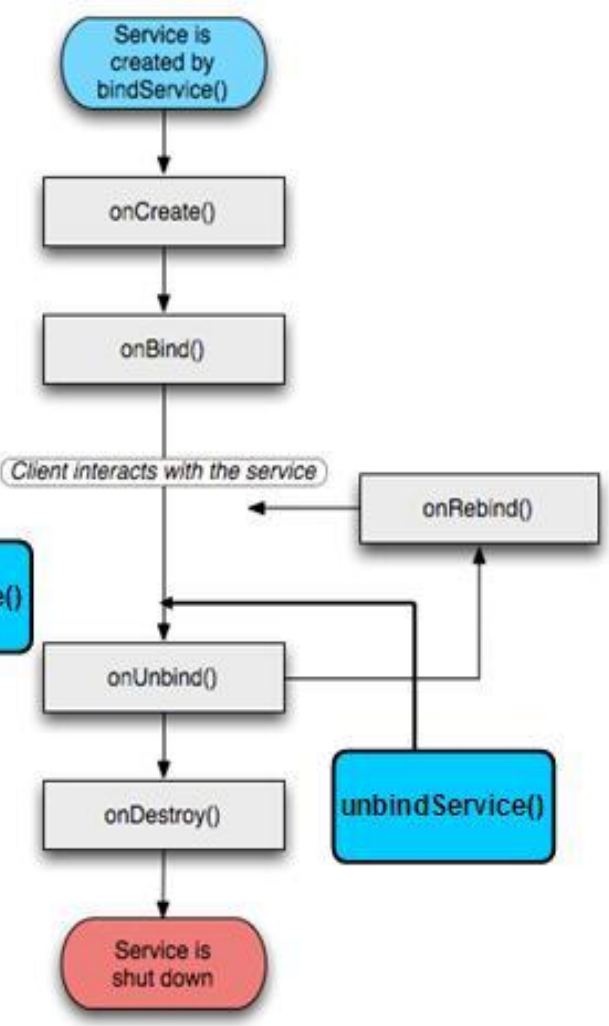
Второй тип: *Подключение*



Какой-то компонент ***подключается*** к сервису с помощью bindService().

Такой вариант позволяет взаимодействовать с сервисом, передавать данные, получать результат. К такому сервису может подключаться сразу несколько компонентов, но он работает до тех пор пока к нему подключен хотя бы один компонент. Когда все компоненты отключатся от сервиса, он будет уничтожен.

Обычно с таким сервисом взаимодействуют с помощью интерфейса, написанного в aidl http://developer.android.com/guide/components/aidl.html. При завершении будет вызван метод onDestroy().



Жизненный цикл сервиса типа Подключение

Как только был вызван метод bindService() система создаст сервис (будет вызван метод onCreate()), затем будет вызван метод onBind() сервиса в котором сервис вернет подключенному компоненту объект IBinder для дальнейшего взаимодействия с этим сервисом. Через IBinder можно вызывать все методы описанные в соответствующем aidl (Android Interface Definition Language).

Сервис может сочетать эти два варианта работы.

Совмещение типов работы сервиса

Можно создать сервис, который одновременно и запущен, и привязан (подключение). Это означает, что сервис можно запустить путем вызова метода `startService()`, который позволяет сервису работать неограниченное время, а также позволяет клиентам привязываться к нему с помощью вызова метода `bindService()`.

Если разрешить запуск и привязку сервиса, то после его запуска система *не* уничтожает его после отмены всех привязок клиентов. Вместо этого необходимо явным образом остановить сервис, вызвав метод `stopSelf()` или `stopService()`.

В большинстве приложений не следует использовать AIDL чтобы создать сервис с подключением, так как это может потребовать создания более сложной архитектуры.

Intent Service

Существует особый вариант сервиса (подкласс) - IntentService.

Класс позволяет обрабатывать присланные запросы (интенты) в другом потоке (в отличии от обычного сервиса обработка таких запросов будет происходить строго в порядке очереди, а не одновременно).
IntentService создает новый поток для своей работы. Затем берет все Intent пришедшие ему в onStartCommand и отправляет их на обработку в этот поток. И далее они поочередно обрабатываются в отдельном потоке методом onHandleIntent. Когда последний Intent из очереди обработан, сервис сам завершает свою работу.

Жизненный цикл Intent Service

Класс должен обязательно содержать конструктор, который возвращает super("название потока для обработки данных"); .

```
public SomeService() {
        super("somename");
}
```

После вызова метода startService() из вызывающего компонента, будет вызван метод onStartCommand(Intent, int, int), который должен всегда возвращать super.onStartCommand(intent, flags,startId); для правильной обработки приходящих интентов. Интенты будут по очереди приходить в метод onHandleIntent(), который необходимо переопределить и указать в нем как обработать приходящие интенты, и автоматически обрабатываться в другом потоке.
Когда все интенты будут обработаны сервис самостоятельно завершится.

Только методы onHandleIntent(), onBind() не требуют вызова super соответствующего метода суперкласса.