4.2. Сервисы в Android. Назначение и виды сервисов.

Сайт: Samsung Innovation Campus
Курс: Мобильная разработка на Kotlin
Книга: 4.2. Сервисы в Android. Назначение и виды сервисов.

Напечатано:: Murad Rezvan
Дата: понедельник, 3 июня 2024, 17:54

Оглавление

1. Сервисы в Android

Сервис в Android — это компонент приложения, который может выполнять длительные операции в фоновом режиме и не содержит пользовательского интерфейса (UI).

Сервисы (службы) в Android представляют собой фоновые процессы. Они не имеют интерфейса пользователя (UI) и работают без его вмешательства. Службы являются компонентами приложения, но способны выполнять действия, даже когда приложение невидимо, пассивно или вовсе закрыто. В отличие от активностей, сервисы предназначены для длительного существования и обладают более высоким приоритетом, чем скрытые активности. Тем самым, вероятность закрытия сервиса системой при нехватке ресурсов намного ниже. К тому же, служба может быть настроена таким образом, что будет автоматически запущена, как только найдутся необходимые ресурсы.

Сегодня, мы можем встретить массу приложений, использующих сервисы. Работая в фоновом режиме, сервисы имеют возможность выполнять сетевые запросы, вызывать уведомления, обрабатывать информацию, проигрывать музыку и выполнять множество иных «закулисных» задач.

Многие компоненты приложения могут запускать сервисы, обмениваться данными и закрывать их. В роли таких компонентов могут выступать как активности, так и другие сервисы. Стоит подчеркнуть, что сервис может быть закрыт как компонентом приложения, так и самостоятельно инициировать собственное завершение.

Обратите внимание, что, несмотря на работу в фоновом режиме, сервисы работают в основном потоке приложения (main UI thread). В случае выполнения «тяжелых» задач их необходимо запускать в отдельном потоке, так как это может привести к «неприятным» задержкам в UI пользователя.

Кроме создания собственных сервисов, можно использовать системные сервисы. Приведем некоторые из них:

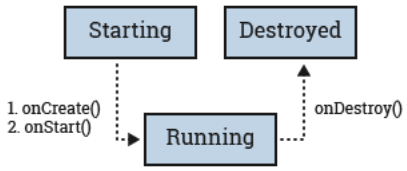
- Account Service — сервис для управления пользовательскими учетными записями;
- Activity Service — сервис для управления активностями;
- Alarm Service — сервис для отправки разовых или периодических оповещений в заданное время;
- Bluetooth Service — сервис для Bluetooth;
- Clipboard Service — сервис для управления буфером обмена;
- Connectivity Service — сервис для управления сетевыми соединениями;
- Download Service — сервис для управления загрузками;
- Input Method Service — сервис для управления текстовым вводом;
- JobScheduler — сервис для планирования задач;
- Location Service — сервис для отслеживания координат;
- Layout Inflater Service — сервис для управления компоновкой экрана при динамическом создании из кода;
- NFC Service — сервис для управления NFC;
- Notification Service — сервис для управления уведомлениями;
- Power Service — сервис для управления энергопотреблением;
- Search Service — сервис для управления глобальным поиском;
- Sensor Service — сервис для доступа к датчикам;
- Telephony Service — сервис для управления телефонными функциями;
- Vibrator Service — сервис для доступа к виброзвонку;
- Wallpaper Service — сервис для управления обоями на домашнем экране;
- Wifi Service — служба для управления соединениями Wi-Fi.

Google последовательно борется с ограничениями для служб, урезая их возможности. Делается это для того, чтобы службы не висели в памяти бесконечно долго и тратили заряд батареи. Ограничение возможностей происходит постепенно от версии к версии. В последних версиях уже можно столкнуться с примерами, когда выкидывается исключение при неправильной работе с службами.

Для решения проблем следует изучить такие вещи как JobScheduler, Firebase Job Dispatcher, WorkManager.

2. Жизненный цикл сервиса

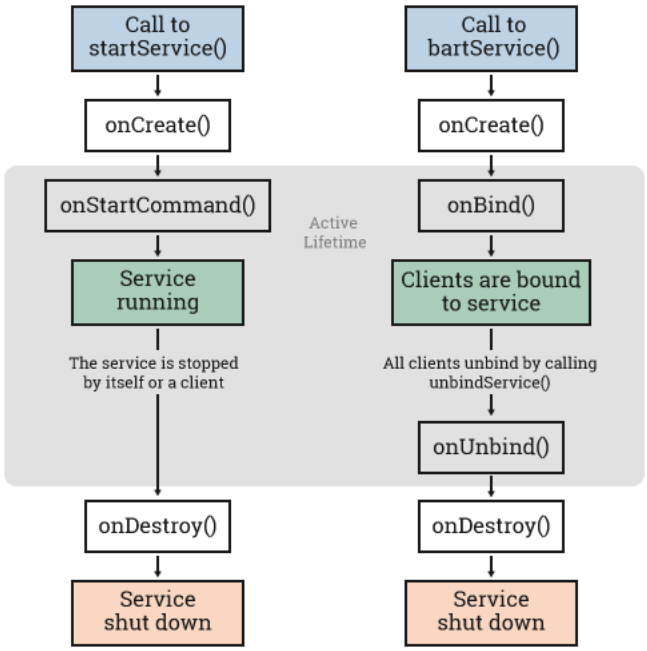
Жизненный цикл сервиса, на первый взгляд, более прост, чем у активностей или фрагментов.



Все сервисы наследуются от класса Service и проходят следующие этапы жизненного цикла:

- `onCreate()` — вызывается при создании сервиса;
- `onStartCommand()` — вызывается при получении сервисом команды, отправленной с помощью метода startService();
- `onBind()` — вызывается при закреплении клиента за сервисом с помощью метода bindService();
- `onDestroy()` — вызывается при завершении работы сервиса.

На самом деле, сервис имеет два вложенных цикла во время своей жизни и два различных варианта поведения.



Сервисы могут работать в двух режимах:

- *Запущенный (started)*. Сервис, запущенный с помощью метода `startService()`. Будучи запущенным, сервис может работать неопределенно долгое время, но чаще служба запускается для выполнение разовой операции, например, загрузки информации по сети. Когда необходимые действия выполнены, сервис должен самостоятельно завершить работу.
- *Привязанный (bound)*. Сервис, к которому другой компонент приложения привязался с помощью метода `bindService()`. Привязанный сервис предоставляет клиент-серверный интерфейс, через который с ним можно взаимодействовать: посылать запросы и получать ответы. Привязанный сервис уничтожается, когда от него отвязывается последний привязанный клиент.

В реальных приложениях службы могут работать и в «смешанном» режиме: быть запущенными с помощью `startService()`, чтобы работать неограниченно долго, в то же время разрешая «привязку» (binding) для взаимодействия с клиентами.

Можно установить подключение к работающей службе и использовать это подключение для взаимодействия со службой. Подключение устанавливают вызовом метода `bindService()`и закрывают вызовом `unbindService()`. Если служба уже была остановлена, вызов метода `bindService()` может ее запустить.

Методы `onCreate()` и `onDestroy()` вызываются для всех сервисов независимо от того, запускаются ли они через `startService()` или `bindService()`.

3. Создание сервиса с помощью класса IntentService.

Как отмечалось ранее, службы по умолчанию запускаются в том же основном потоке, что и компонент, из которого они запускаются. Таким образом, любые задачи с интенсивным использованием ЦП, которые должны выполняться службой, должны выполняться в новом потоке.

Класс `IntentService` - это вспомогательный класс (унаследованный от класса `Service`), который создает рабочий поток для обработки фоновых задач и обрабатывает каждый запрос асинхронным образом. После того, как служба обработала все запросы в очереди, она просто завершает работу. Все, что требуется при использовании класса `IntentService`, - это реализовать метод `onHandleIntent()`, содержащий код, который будет выполняться для каждого запроса. Для сервисов, не требующих синхронной обработки запросов, рекомендуется использовать `IntentService`. Однако службы, требующие синхронной обработки запросов, должны будут создать подкласс от класса `Service` и вручную реализовать и управлять потоками для эффективной обработки любых задач, интенсивно использующих ЦП.

Чтобы сервис можно было использовать, он должен быть сначала объявлен в файле манифеста. Как минимум, элемент `<service>` должен содержать свойство, объявляющее имя класса службы, как показано в следующем фрагменте XML:

```
<application
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name" >
    <activity
        android:label="@string/app_name"
        android:name=".TestActivity" >
        <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
        <service android:name="MyService>
        </service>
</application>
```

По умолчанию службы объявляются общедоступными, поскольку к ним могут обращаться компоненты вне пакета приложения, в котором они находятся. Чтобы сделать службу приватной, свойство android: exported должно быть объявлено как `false` в элементе `<service>` файла манифеста. Например:

```
<service android:name="MyService"
    android:exported="false">
</service>
```

Чтобы заставить службу запускаться в ее собственном процессе, добавьте свойство `android:process` к элементу `<service>`, объявив имя процесса с префиксом двоеточия (:):

```
<service android:name="MyService"
    android:exported="false"
    android:process=":myprocess">
</service>
```

Префикс двоеточия указывает, что новый процесс является частным для локального приложения. Однако, если имя процесса начинается со строчной буквы вместо двоеточия, процесс будет глобальным и доступен для использования другими компонентами.

Пример

Рассмотрим пример создания `IntentService`.

Создадим класс `MyIntentService` наследник класса `IntentService`. При создании подкласса класса необходимо соблюдать два правила. Во-первых, должен быть реализован конструктор класса, который вызывает конструктор суперкласса, передавая имя класса службы. Во-вторых, класс должен переопределить метод `onHandleIntent()`. Класс должен выглядеть следующим образом:

```
import android.app.IntentService
import android.content.Intent

class MyIntentService : IntentService("MyIntentService") {
         override fun onHandleIntent(arg0: Intent?) {
    }
}
```

Все, что остается на этом этапе, - это реализовать некоторый код в методе `onHandleIntent()`, чтобы служба что-то делала при вызове. Обычно это связано с выполнением задачи, на выполнение которой требуется некоторое время, такой как загрузка большого файла или воспроизведение звука. Однако для целей этого примера просто выведем сообщение в Logcat:

```
import android.app.IntentService
import android.content.Intent
import android.util.Log

class MyIntentService : IntentService("MyIntentService") {

    private val TAG = "ServiceExample"

    override fun onHandleIntent(arg0: Intent?) {
        Log.i(TAG, "Intent Service started")
    }
}
```

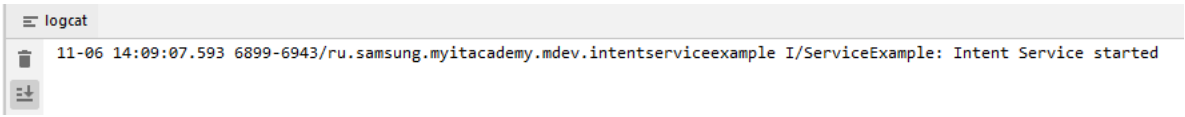
Прежде чем службу можно будет вызвать, сначала ее необходимо добавить в файл манифеста. Для нашего примера это можно сделать следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="ru.samsung.itacademy.mdev.intentserviceexample">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".ServiceExampleActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name=".MyIntentService" />
    </application>

</manifest>
```

Теперь, когда служба реализована и объявлена в файле манифеста, следующим шагом будет добавление кода для запуска службы. Это можно сделать в методе `onCreate()` класса активности следующим образом:

```
package ru.samsung.itacademy.mdev.intentserviceexample

import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.content.Intent

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val intent = Intent(this, MyIntentService::class.java)
        startService(intent)
    }
}
```

Мы создали объект `Intent` с именем класса сервиса для запуска и затем используем его в качестве аргумента метода `startService()`. После запуска приложения и фильтрации по тэгу "ServiceExample" в Logcat появляется сообщение «Intent Service Started».



При использовании Android Studio для создания сервисов можно использовать пункты меню *New -> Service -> Service*. Это позволит автоматически создать класс наследник Service и задекларировать его в манифесте приложения.

4. Создание сервиса с помощью класса Service

Не смотря на то, что использование класса `IntentService` позволяет реализовать сервис с минимизацией количества кода, бывают случаи, когда необходима гибкость класса `Service`. Во избежание одновременного введения слишком большого количества правил, связанных с выполнением трудоемких служебных задач в том же потоке, в данном параграфе приведем пример создания службы с использованием класса Service в основной потоке приложения.

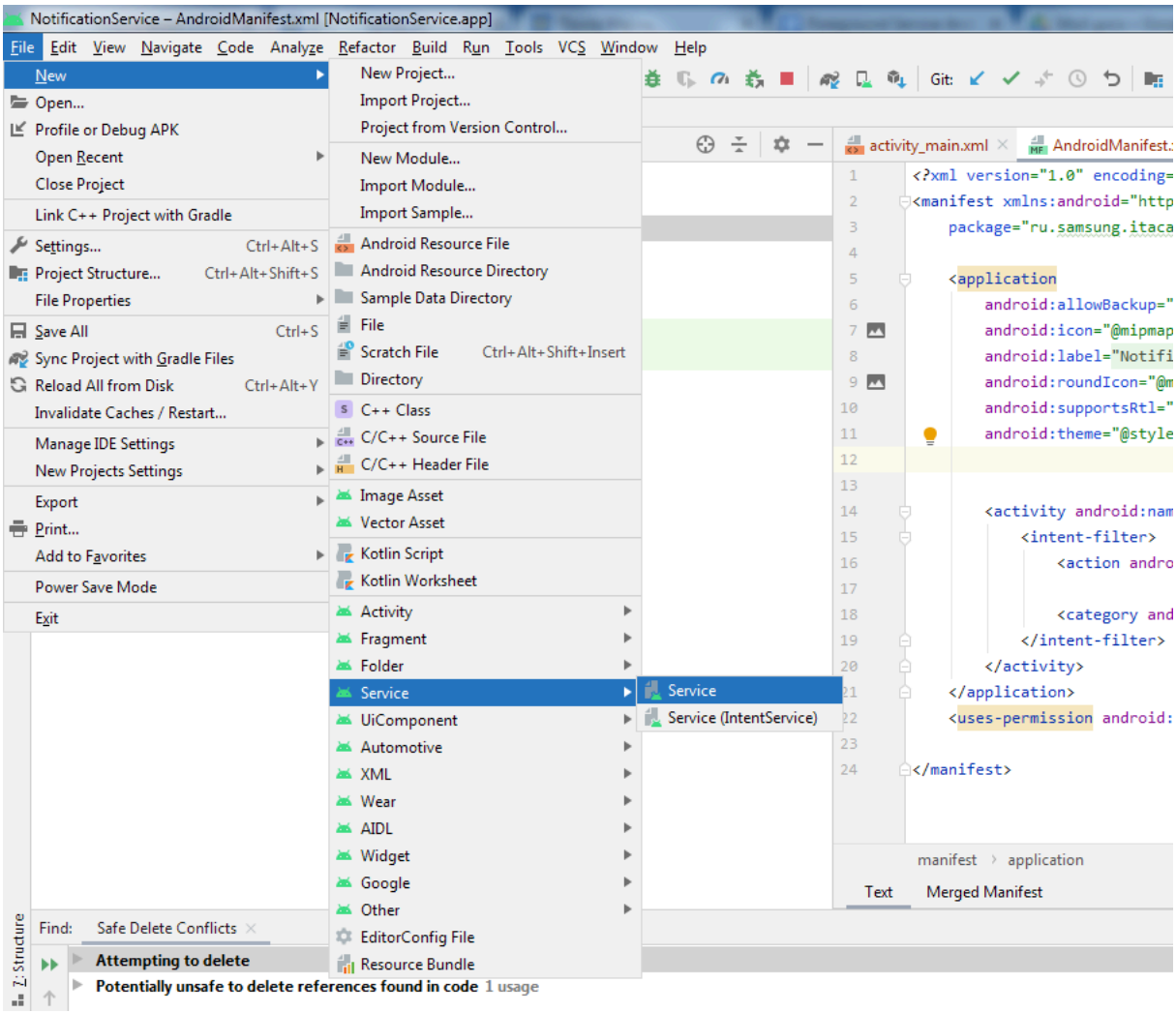
Создадим новый проект и добавим в него новый класс `MyService`, который будет производным от класса `Service`. Минимальным требованием для создания работающей службы является реализация в ней метода `onStartCommand()`, который будет вызываться при запуске службы. Кроме того, метод `onBind()` должен возвращать `null`, чтобы указать системе Android, что это не привязанная служба. В примере ниже метод `onStartCommand()` будет выполнять цикл три раза и каждый раз ждать по по 10 секунд, имитируя выполнение трудоемкой операции. Также для нашего сервиса реализуем методы `onCreate()` и `onDestroy()`. В них будем выводить соответствующую информацию в лог.

```
package ru.samsung.itacademy.mdev.serviceexample

import android.app.Service
import android.content.Intent
import android.os.IBinder
import android.util.Log

class MyService : Service() {
    private val TAG = "ServiceExample"

    override fun onCreate() {
        Log.i(TAG, "Service onCreate")
    }

    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int): Int {
        Log.i(TAG, "Service onStartCommand " + startId)
        var i: Int = 0
        while (i <= 3) {
            try {
                Thread.sleep(10000)
                i++
            } catch (e: Exception) { }
            Log.i(TAG, "Service running")
        }
        return Service.START_STICKY
    }

    override fun onBind(intent: Intent): IBinder? {
        Log.i(TAG, "Service onBind")
        return null
    }

    override fun onDestroy() {
        Log.i(TAG, "Service onDestroy")
    }
}
```

После реализации службы откройте файл AndroidManifest.xml и добавьте информацию о нашем сервисе.

```
<service
 android:name=".MyService"
            android:enabled="true"
            android:exported="true" >
</service>
```

В Android Studio существует возможность создавать службы автоматически. Для этого нажмите *File -> New -> Service*.



Далее в поле Class Name необходимо ввести имя файла для сервиса.

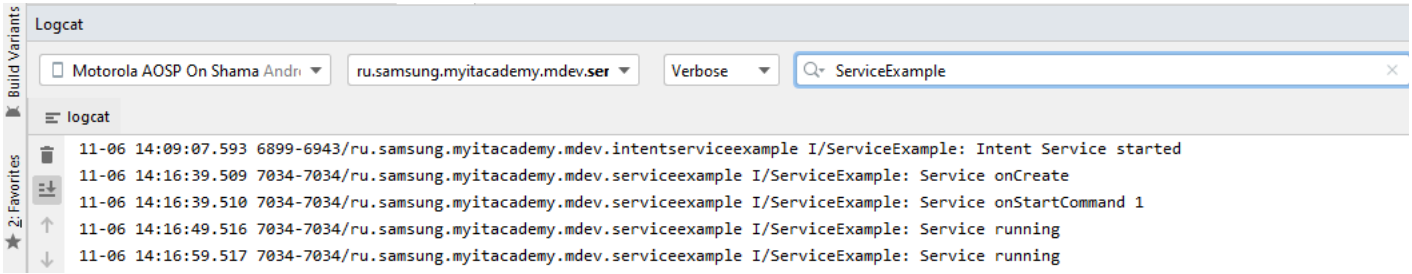
Отсутствие отдельного потока для сервиса создает проблему удобства его использования. Добавим кнопку в наш пользовательский интерфейс и назначим ей атрибут `onClick` с именем `buttonClick`.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Button"
        tools:layout_editor_absoluteX="3dp"
        tools:layout_editor_absoluteY="16dp"
        android:onClick="buttonClick"/>
</LinearLayout>
```

Далее отредактируем файл MainActivity.kt, добавив в него метод `buttonClick()`:

```
package ru.samsung.itacademy.mdev.serviceexample

import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.content.Intent
import android.view.View

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

    }

    fun buttonClick(view: View)
    {
        intent = Intent(this, MyService::class.java)
        startService(intent)
    }
}
```

Метод `buttonClick()` создает объект намерения для новой службы, а затем запускает его. Запустите приложение и после загрузки нажмите созданную кнопку. В Logcat появятся сообщения, указывающие, что был вызван метод `onCreate()` и что в методе `onStartCommand()` выполняется цикл.



Прежде чем появится последнее сообщение цикла, попробуйте второй раз нажать кнопку. Обратите внимание, что UI не отвечает. Это происходит потому, что основной поток приложения занят службой, пока он выполняет цикл. Очевидно, что код службы необходимо изменить, чтобы задачи выполнялись в отдельном потоке.