4.3. Сервисы в Android. Bound сервис

Сайт: Samsung Innovation Campus
Курс: Мобильная разработка на Kotlin
Книга: 4.3. Сервисы в Android. Bound сервис

Напечатано:: Murad Rezvan
Дата: понедельник, 3 июня 2024, 17:55

Оглавление

1. Связанная служба (Bound service)

Привязанная служба (Bound service) предоставляет интерфейс типа клиент-сервер. Привязанная служба позволяет компонентам приложения взаимодействовать со службой, отправлять запросы, получать результаты и даже делать то же самое с другими процессами через IPC. Привязанная служба обычно работает, пока другой компонент приложения привязан к ней. Она не работает постоянно в фоновом режиме. Привязанная служба очень похожа на уже запущенную службу за одним исключением: запущенная служба обычно не возвращает результаты и не разрешает взаимодействие с компонентом, из которого она была запущена. При использовании Bound-сервисов возможно взаимодействовать компонентов, запустивших сервис с самим сервисом и получение результатов от него. Это взаимодействие также может происходить между процессами.

Например, в активности можно запустить службу для обработки звука, при этом интерфейс активности скорее всего будет содержать элементы управления для приостановки воспроизведения или перехода к следующему треку. Точно так же службе, скорее всего, потребуется передать информацию вызывающей ее активности, чтобы указать, что текущая звуковая дорожка завершена, и предоставить сведения о следующей дорожке, которая вот-вот начнется.

Активность или фрагмент (также называемый в этом контексте клиентом) запускается и связывается со службой через метод `bindService()`. Кроме того, несколько компонентов могут одновременно подключаться к службе. Когда привязка службы больше не требуется клиенту, следует вызвать метод `unbindService()`. Когда последний связанный клиент отключается от службы, служба будет прекращена системой Android. Важно помнить, что привязанная служба также может быть запущена с помощью вызова `startService()`. После запуска компоненты приложения могут связываться с ней с помощью метода `bindService()`. Когда связанная служба запускается с помощью вызова `startService()`, она будет продолжать свою работу даже после того, как последний клиент "отвяжется" от нее.

В привязанной службе должен быть реализован метод `onBind()`, который вызывается как при первоначальном создании службы, так и при последующей привязке других клиентов к работающей службе. Цель этого метода - вернуть привязанным клиентам объект типа `IBinder`, содержащий информацию, необходимую клиенту для связи со службой.

С точки зрения реализации связи между клиентом и привязанной службой рекомендуемый метод зависит от того, находятся ли клиент и служба в одном или разных процессах и является ли служба приватной для клиента. Локальная связь может быть достигнута путем расширения класса `Binder` и возврата экземпляра из метода `onBind()`.

2. Варианты взаимодействия связанных сервисов

Как и в случае с обычными службами, привязанные службы, позволяют приложениям выполнять задачи в фоновом режиме. Однако, в отличие от запущенных служб, несколько клиентских компонентов (активностей, фрагментов и т.п.) могут связываться с привязанной службой и после связывания взаимодействовать с ней, используя множество различных механизмов.

Связанные службы создаются как подклассы класса `Service`, в них должен быть реализован метод `onBind()`. Клиентские компоненты связываются со службой посредством вызова метода `bindService()`. Первый запрос привязки к привязанной службе приведет к вызову метода `onBind()` этой службы (последующие запросы привязки не запускают вызов `onBind()`). Клиенты, желающие выполнить привязку к службе, также должны реализовать интерфейс `ServiceConnection`, содержащий методы `onServiceConnected()` и `onServiceDisconnected()`, которые будут вызываться, когда соединение клиент-сервер будет установлено или отключено соответственно. В случае метода `onServiceConnected()` ему будет передан объект `IBinder`, содержащий информацию, необходимую клиенту для взаимодействия со службой.

Существует два рекомендуемых механизма для реализации взаимодействия между клиентскими компонентами и привязанным сервисом. В случае, если привязанная служба является локальной и частной для приложения (она работает в том же процессе и недоступна для компонентов в других приложениях), рекомендуется создать подкласс `Binder` и расширите его, чтобы обеспечить интерфейс для службы. Затем экземпляр этого объекта `Binder` возвращается методом `onBind()` и впоследствии используется клиентским компонентом для прямого доступа к методам и данным, хранящимся в службе. В ситуациях, когда привязанная служба не является локальной для приложения (другими словами, она выполняется в процессе, отличном от клиентского компонента), взаимодействие лучше всего достигается с помощью реализации `Messenger / Handler`. В оставшейся части будет создан пример с целью демонстрации действий, необходимых для создания, запуска и организации взаимодействия с локальной частной службой.

3. Пример локальной привязанной службы в Android

Разработаем приложение, которое будет состоять из одной активности и привязанной службы. Назначение службы - получать текущее время от системы и передать эту информацию в активность, где данные будут отображаться пользователю. Запустим Android Studio и выполним обычные шаги по созданию проекта. Назовем проект LocalBound. Добавим в проект класс BoundService - привязанную службу, который будет выглядеть следующим образом:

```
package ru.samsung.itacademy.mdev.localbound

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

class BoundService : Service() {

    override fun onBind(intent: Intent): IBinder? {
         throw UnsupportedOperationException("Not yet implemented")
    }
}
```

Как было отмечено ранее, локальные привязанные службы могут взаимодействовать с клиентами, передавая им объект `Binder`. Для этого необходимо создать подкласс класса `Binder` внутри класса службы и его добавить в него методы, которые могут быть вызваны клиентом. В большинстве случаев он просто включает реализацию метода, который возвращает ссылку на связанный экземпляр службы. Далее клиент может затем напрямую обращаться к данным и вызывать методы в привязанной службе. Следовательно, для нам требуется внести некоторые изменения в класс `BoundService`. В первую очередь необходимо объявить подкласс `Binder`, который будет содержать единственный метод с именем `getService()`, возвращающий ссылку на текущий экземпляр объекта сервиса. Теперь файл BoundService.kt может выглядеть следующим образом:

```
package ru.samsung.itacademy.mdev.localbound

import android.app.Service
import android.content.Intent
import android.os.IBinder
import android.os.Binder

class BoundService : Service() {

    private val myBinder = MyLocalBinder()

    override fun onBind(intent: Intent): IBinder? {
        throw UnsupportedOperationException("Not yet implemented")
    }

    inner class MyLocalBinder : Binder() {
        fun getService() : BoundService {
            return this@BoundService
        }
    }
}
```

Далее изменим метод `onBind()`, так чтобы он возвращал ссылку на объект `myBinder`, и новый добавим метод `getCurrentTime()` для возврата текущего времени:

```
package ru.samsung.itacademy.mdev.localbound

import android.app.Service
import android.content.Intent
import android.os.Binder
import android.os.IBinder
import java.text.SimpleDateFormat
import java.util.*

class BoundService : Service() {

    private val myBinder = MyLocalBinder()

    override fun onBind(intent: Intent): IBinder? {
        return myBinder
    }

    fun getCurrentTime(): String {
        val dateformat = SimpleDateFormat("HH:mm:ss MM/dd/yyyy",
                Locale.US)
        return dateformat.format(Date())
    }

    inner class MyLocalBinder : Binder() {
        fun getService() : BoundService {
            return this@BoundService
        }

    }
}
```

Теперь службу необходимо добавить в файл AndroidManifest.xml:

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

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

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

        <service
            android:name=".BoundService"
            android:enabled="true"
            android:exported="true" >
        </service>
    </application>

</manifest>
```

Следующий этап - написание кода активности, реализация ее привязки к службе и вызова метода `getCurrentTime()`.

Как отмечалось ранее, для успешного связывания активности со службой и получения объекта `IBinder`, возвращаемого методом `onBind()`, требуется создать подкласс `ServiceConnection` и реализовать в нем методы `onServiceConnected()` и `onServiceDisconnected()`. Изменим файл *MainActivity.kt* следующим образом:

```
package ru.samsung.itacademy.mdev.localbound

import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.content.ComponentName
import android.content.Context
import android.content.ServiceConnection
import android.os.IBinder
import android.content.Intent

class MainActivity : AppCompatActivity() {

    var myService: BoundService? = null
    var isBound = false

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

    private val myConnection = object : ServiceConnection {
        override fun onServiceConnected(className: ComponentName,
                                        service: IBinder) {
            val binder = service as BoundService.MyLocalBinder
            myService = binder.getService()
            isBound = true
        }

        override fun onServiceDisconnected(name: ComponentName) {
            isBound = false
        }
    }
}
```

Метод `onServiceConnected()` будет вызываться, когда активность успешно привяжется к службе. Метод передается в качестве аргумента объекту `IBinder`, возвращаемому методом `onBind()`. Этот аргумент приводится к объекту типа `MyLocalBinder`, а затем вызывается метод `getService()` объекта связывания для получения ссылки на экземпляр сервиса. Переменная `isBound` используется, чтобы указать, что соединение было успешно установлено. Метод `onServiceDisconnected()` вызывается при завершении соединения и просто устанавливает `isBound` значение `false`.

После установки соединения нужно изменить код активности для ее привязки к сервису. Для этого необходимо создать намерение и вызвать метод `bindService()`, который может быть выполнен в методе `onCreate ()`:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    val intent = Intent(this, BoundService::class.java)
    bindService(intent, myConnection, Context.BIND_AUTO_CREATE)
}
```

Осталось реализовать механизм вызова метода `getCurrentTime()` и отображения результата. Добавим в файл приложения *activity_main.xml* компонент `TextView` с идентификатором `myTextView` и кнопку с надписью «Show Time» и атрибутом `onClick` со значением `showTime`:

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

    <TextView
        android:text="Time"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true"
        android:id="@+id/myTextView" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Show time"
        android:id="@+id/button"
        android:layout_below="@+id/myTextView"
        android:layout_centerHorizontal="true"
        android:onClick="showTime"
        android:layout_marginTop="44dp" />

</RelativeLayout>
```

Внешний вид активности будет следующим:

Наконец, отредактируем код в файле *MainActivity.kt*, в котором реализуем метод `showTime()`, который просто вызывает метод `getCurrentTime()` службы (метод доступен изнутри активности через ссылку myService благодаря методу `onServiceConnected()`) и запишем полученную строку `TextView`. Полностью код *MainActivity.kt* будет выглядеть следующим образом:

```
package ru.samsung.itacademy.mdev.localbound

import android.content.ComponentName
import android.content.Context
import android.content.ServiceConnection
import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.os.IBinder
import android.content.Intent
import android.view.View
import kotlinx.android.synthetic.main.activity_local_bound.*

class MainActivity: AppCompatActivity() {

    var myService: BoundService? = null
    var isBound = false

    fun showTime(view: View) {
        val currentTime = myService?.getCurrentTime()
        myTextView.text = currentTime
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val intent = Intent(this, BoundService::class.java)
        bindService(intent, myConnection, Context.BIND_AUTO_CREATE)
    }

    private val myConnection = object : ServiceConnection {
        override fun onServiceConnected(className: ComponentName,
                                        service: IBinder) {
            val binder = service as BoundService.MyLocalBinder
            myService = binder.getService()
            isBound = true
        }

        override fun onServiceDisconnected(name: ComponentName) {
            isBound = false
        }
    }
}
```

Запустим приложение и нажмем кнопку. Обратите внимание, что текстовое представление изменится и отобразит текущую дату и время.


14:53
LocalBound
14:50:44 11/06/2020
SHOW TIME

4. Пример удаленной привязанной службы в Android

Как показано в предыдущем параграфе, взаимодействие между клиентом и локальной службой может быть реализовано путем возврата клиенту объекта `IBinder`, содержащего ссылку на объект сервиса. Однако в случае удаленных служб данный подход не работает, потому что удаленная служба работает в другом процессе и не может быть доступна напрямую от клиента.

В случае удаленных служб необходимо создать конфигурацию `Messenger` и `Handler`, которая позволяет передавать сообщения через процессы между клиентом и службой.

В частности, служба создает экземпляр класса `Handler` будет вызываться при получении сообщения от клиента. Задача `Handler` - создать объект `Messenger`, который, в свою очередь, создаст объект `IBinder`, возвращенный клиенту в методе `onBind()`. `IBinder` используется клиентом для создания экземпляра объекта класса `Messenger` и, впоследствии, для отправки сообщений обработчику сервиса. Каждый раз, когда клиент отправляет сообщение, вызывается метод `handleMessage()`. Приведем простой пример из одной активности и службы, выполняемых в отдельных процессах. Механизм `Messenger / Handler` будет использоваться для отправки строки службе, которая затем отобразит эту строку `Toast`.

Создадим проект, а в нем файл сервиса RemoteService.kt:

```
package ru.samsung.myakademy.remotebound

import android.app.Service
import android.content.Intent
import android.os.Handler
import android.os.IBinder
import android.os.Handler
import android.os.Message
import android.os.Messenger
import android.widget.Toast

class RemoteService: Service() {

    inner class IncomingHandler : Handler() {
        override fun handleMessage(msg: Message) {

            val data = msg.data
            val dataString = data.getString("MyString")
            Toast.makeText(applicationContext,
                    dataString, Toast.LENGTH_SHORT).show()
        }
    }

    override fun onBind(intent: Intent): IBinder? {
          throw UnsupportedOperationException("Not yet implemented")
    }
}
```

Изменим метод `onBind()` таким образом, чтобы он возвращал объект `IBinder`, содержащий объект `Messenger`:

```
private val myMessenger = Messenger(IncomingHandler())

override fun onBind(intent: Intent): IBinder? {
    return myMessenger.binder
}
```

Первая строка приведенного выше кода создает новый экземпляр класса обработчика и передает его конструктору объекта `Messenger`. Внутри метода `onBind()` вызывается метод `getBinder()` объекта мессенджера, чтобы вернуть объект `IBinder` мессенджера.

Для установки взаимодействия между клиентом и удаленной службой, требуется настроить службу для запуска в отдельном процессе от остальной части приложения. Это можно сделать путем добавления свойства `android:process` в тег `<service>` в файле манифеста. AndroidManifest.xml будет выглядеть следующим образом:

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

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

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

        <service
            android:name=".RemoteService"
            android:enabled="true"
            android:exported="true"
            android:process=":my_process" >
        </service>
    </service>
    </application>

</manifest>
```

Как и в случае с локальной службой, в клиенте должен быть реализовать экземпляр класса `ServiceConnection` с методами `onServiceConnected()` и `onServiceDisconnected()`. Также, как и в случае с локальными службами, методу `onServiceConnected()` будет передан объект `IBinder`, возвращенный методом `onBind()` удаленной службы, который будет использоваться для отправки сообщений обработчику сервера. Изменим код *MainActivity.kt* следующим образом:

```
package ru.samsung.myakademy.remotebound

import android.support.v7.app.AppCompatActivity
import android.content.ComponentName
import android.content.ServiceConnection
import android.os.*
import android.view.View

class MainActivity : AppCompatActivity() {

    var myService: Messenger? = null
    var isBound: Boolean = false

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

    private val myConnection = object : ServiceConnection {
        override fun onServiceConnected(
                className: ComponentName,
                service: IBinder) {
            myService = Messenger(service)
            isBound = true
        }

        override fun onServiceDisconnected(
                className: ComponentName) {
            myService = null
            isBound = false
        }
    }
}
```

Затем необходимо добавить код для привязки к удаленной службе. Перепишем метод onCreate() следующим образом:

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

    val intent = Intent(getApplicationContext(), RemoteService::class.java)
    bindService(intent, myConnection, Context.BIND_AUTO_CREATE)
}
```

Разметка главной активности будет состоять из одной кнопки с атрибутом `onClick` со значением `sendMessage`.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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="match_parent"
        android:layout_height="wrap_content"
        android:text="Show message"
        android:onClick="sendMessage"
        android:textSize="30sp"
        tools:layout_editor_absoluteX="25dp"
        tools:layout_editor_absoluteY="32dp" />
</RelativeLayout>
```

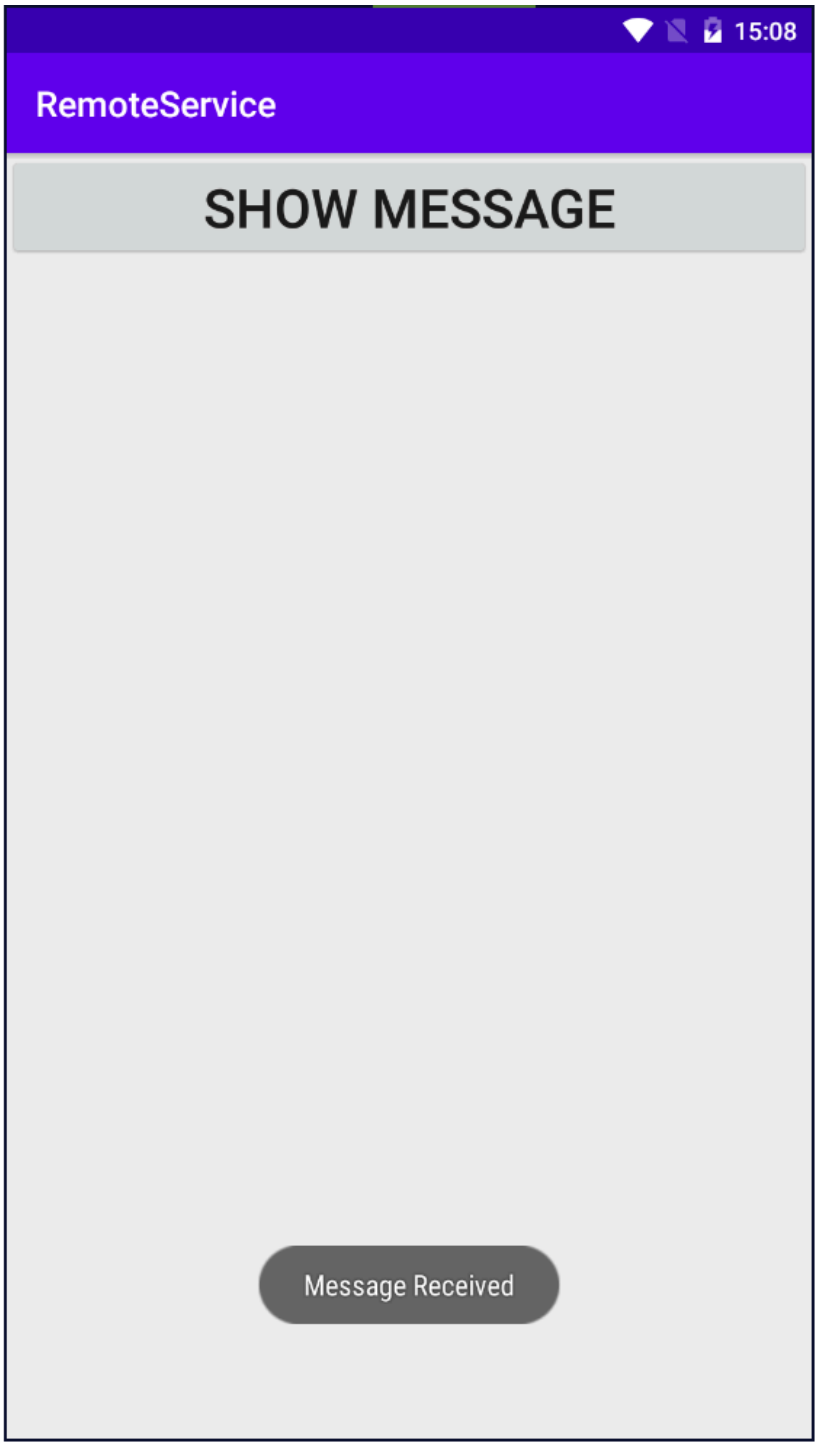
Осталось реализовать метод `sendMessage()` в классе MainActivity, который вызывается, когда пользователь нажимает на кнопку. Этот метод проверяет, подключен ли сервис, создает, содержащий строку сообщения, добавляет его в объект сообщения и отправляет его на сервер:

```
fun sendMessage(view: View) {
     if (!isBound) return
     val msg = Message.obtain()

    val bundle = Bundle()
    bundle.putString("MyString", "Message Received")

    msg.data = bundle
    try {
        myService?.send(msg)
    } catch (e: RemoteException) {
        e.printStackTrace()
    }
}
```

После запуска приложения и нажатия на кнопку в пользовательском интерфейсе появляется Toast-сообщение с надписью "Message Received".



Начать тур для пользователя на этой странице