13. 서비스 컴포넌트

1. 서비스 이해하기
2. 바인딩 서비스
3. 백그라운드 제약
4. 잡 스케줄러
5. MP3 재생 앱 만들기

1. 서비스 이해하기

- 서비스 생성과 실행
  - 서비스 컴포넌트는 Service 클래스를 상속받아서 작성
  - 서비스도 컴포넌트이므로 매니페스트에 등록

• 서비스 컴포넌트 생성

```
class MyService : Service() {
    override fun onBind(intent: Intent): IBinder? {
        return null
    }
}
```

• 서비스 컴포넌트 등록

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

1. 서비스 이해하기

- startService() 함수로 실행
  - startService() 함수로 서비스를 실행하려면 해당 서비스를 인텐트에 담아서 매개변수로 전달
  - 외부 앱의 서비스라면 암시적 인텐트로 실행해야 하므로 setPackage() 함수를 이용해 앱의 패키지명을 명시
  - 서비스를 종료하려면 stopService() 함수로 인텐트를 전달

• 서비스 실행

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

• 암시적 인텐트로 실행

```
val intent = Intent("ACTION_OUTER_SERVICE")
intent.setPackage("com.example.test_outter")
startService(intent)
```

• 서비스 종료

```
val intent = Intent(this, MyService::class.java)
stopService(intent)
```

1. 서비스 이해하기

- bindService() 함수로 실행
  - ServiceConnection 인터페이스를 구현한 객체를 준비
  - onServiceConnected()는 bindService() 함수로 서비스를 구동할 때 자동으로 호출
  - nServiceDisconnected()는 unbindService() 함수로 서비스를 종료할 때 자동으로 호출

• ServiceConnection 인터페이스 구현

```
val connection: ServiceConnection = object : ServiceConnection {
    override fun onServiceConnected(name: ComponentName?, service: IBinder?) { }
    override fun onServiceDisconnected(name: ComponentName?) { }
}
```

• 서비스 실행

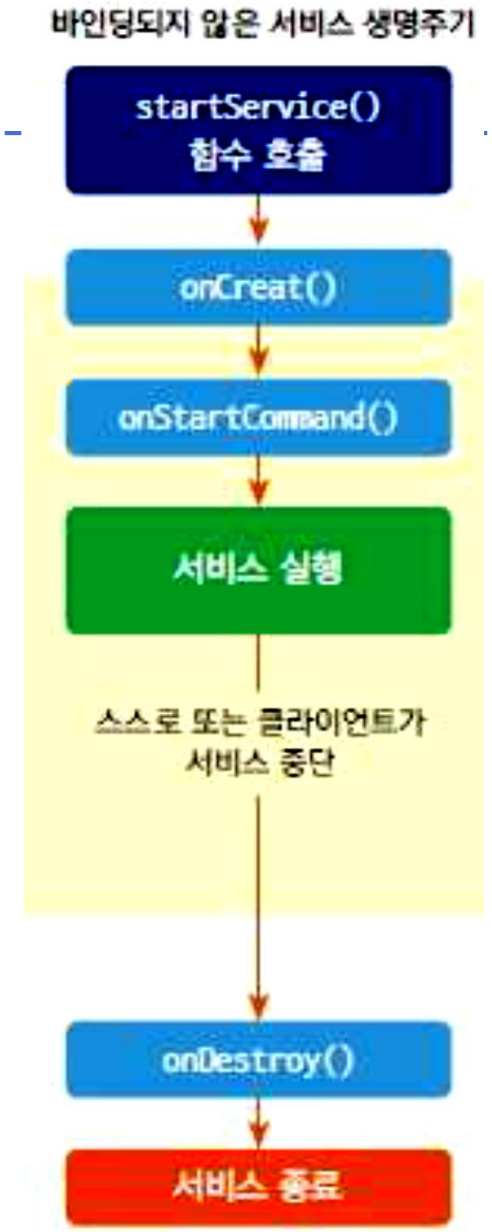
```
val intent = Intent(this, MyService::class.java)
bindService(intent, connection, Context.BIND_AUTO_CREATE)
```

• 서비스 종료

```
unbindService(connection)
```

1. 서비스 이해하기

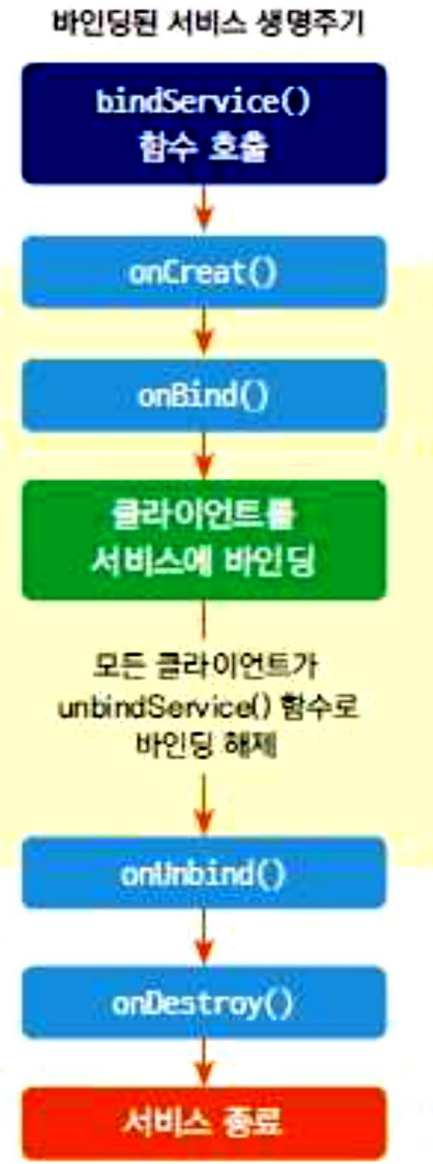
- **서비스 생명주기**
  - 서비스를 실행하는 2가지 방법은 startService()와 bindService()이므로
  - 어느 함수를 이용해 서비스를 실행하는지에 따라 생명주기가 나뉩니다.
  - startService() 함수에서 서비스 객체를 생성하면 onCreate()
    →onStartCommand() 함수가 호출
  - onCreate() 함수는 서비스 객체가 생성될 때 처음에 한 번만 호출
  - onStartCommand() 함수는 startSer vice() 함수가 실행될 때마다 반복해서 호출
  - stopService() 함수로 서비스가 종료되면 바로 전에 onDestory() 함수가 호출



1. 서비스 이해하기

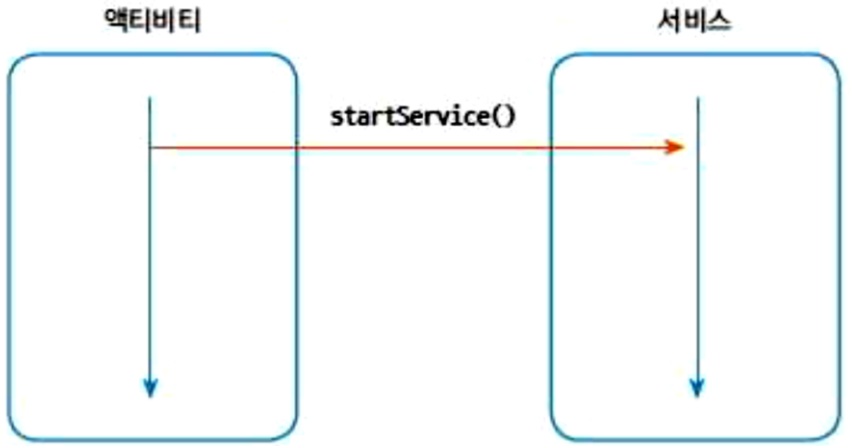
바인딩된 서비스 생명주기

- bindService() 함수에서 서비스 객체를 생성하면 onCreate → onBind() 함수가 호출
- 다시 bindService() 함수로 실행하면 onBind() 함수만 다시 호출
- unbindService() 함수로 서비스를 종료하면 onUnbind() → onDestory() 함수까지 실행

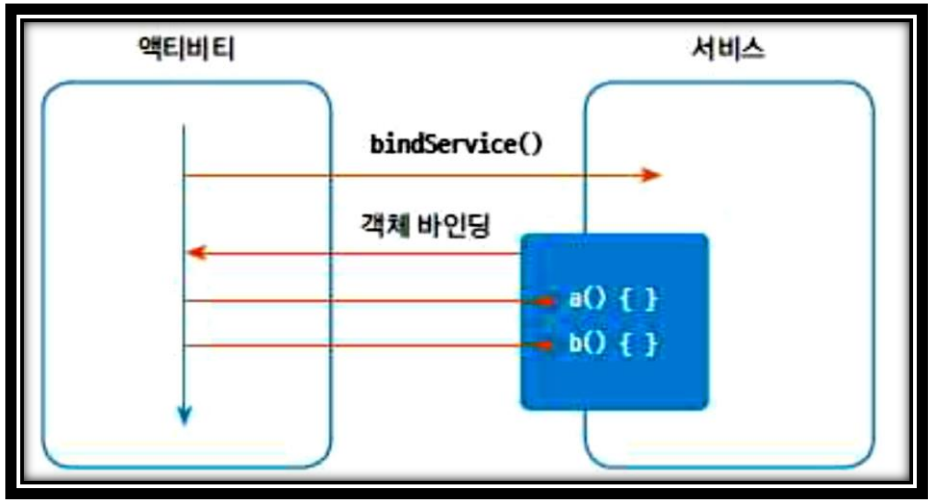


2. 바인딩 서비스

- IBinder 객체 바인딩
  - startService() 함수로 서비스를 실행했다고 가정



  - 서비스와 액티비티가 상호작용 해야 할 때 bindService()



2. 바인딩 서비스

- 서비스 코드
  - onBind() 함수의 반환 타입은 IBinder 인터페이스
  - 서비스를 실행한 곳에서 이 클래스의 함수를 호출하면서 매개변수와 반환값으로 데이터를 주고받음

• 서비스에서 객체 바인딩

```
class MyBinder : Binder() {
    fun funA(arg: Int) {
    }
    fun funB(arg: Int): Int {
        return arg * arg
    }
}
override fun onBind(intent: Intent): IBinder? {
    return MyBinder()
}
```

2. 바인딩 서비스

- 액티비티 코드
- onBind() 함수에서 반환한 객체를 ServiceConnection 인터페이스를 구현한 객체의 onServiceConnected() 함수로 받을 수 있음

• 액티비티에서 객체를 전달받아 사용

```
val connection: ServiceConnection = object : ServiceConnection {
    override fun onServiceConnected(name: ComponentName?, service: IBinder?) {
        servideBinder = service as MyService.MyBinder
    }
    override fun onServiceDisconnected(name: ComponentName?) {
    }
}
```

2. 바인딩 서버스

- 메신저 바인딩
  - API에서 제공하는 Messenger 객체를 바인딩하는 방법
  - Messenger 객체를 이용하는 방법은 프로세스 간 통신할 때도 사용
  - handleMessage() 함수는 외부에서 서비스에 데이터를 전달할 때 자동으로 호출
  - 전달한 데이터는 Message 타입
  - Message의 what값으로는 어떤 성격의 데이터인지를 구분하며 obj 속성으로는 전달된 데이터를 가져옴
  - onBind() 함수의 반환값으로 Messenger 객체를 생성하면서 생성자 매개변수로 Handler를 구현한 객체를 지정

2. 바인딩 서비스

• 메신저 객체를 이용하는 서비스 코드

```
class MyService : Service() {
    lateinit var messenger: Messenger
    internal class IncomingHandler(
        context: Context,
        private val applicationContext: Context = context.applicationContext
    ) : Handler(Looper.getMainLooper()) {
        override fun handleMessage(msg: Message) {
            when (msg.what) {
                10 ->
                    Toast.makeText(applicationContext, "${msg.obj}",
                                   Toast.LENGTH_SHORT).show()
                20 ->
                    Toast.makeText(applicationContext, "${msg.obj}",
                                   Toast.LENGTH_SHORT).show()
                else -> super.handleMessage(msg)
            }
        }
    }
```

```
    override fun onBind(intent: Intent): IBinder? {
        messenger = Messenger(IncomingHandler(this))
        return messenger.binder
    }
}
```

2. 바인딩 서비스

- 액티비티 코드
  - onService Connected() 함수의 매개변수로 넘어온 객체를 Messenger의 생성자 매개변수에 지정

• 메신저 객체를 이용하는 액티비티 코드

```
class MainActivity : AppCompatActivity() {
    lateinit var messenger: Messenger
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        (... 생략 ...)
        val intent = Intent(this, MyService::class.java)
        bindService(intent, connection, Context.BIND_AUTO_CREATE)
    }
    (... 생략 ...)
    val connection: ServiceConnection = object : ServiceConnection {
        override fun onServiceConnected(name: ComponentName?, service: IBinder?) {
            messenger = Messenger(service)
        }
        override fun onServiceDisconnected(name: ComponentName?) {
        }
    }
}
```

• 서비스에 데이터 전달

```
val msg = Message()
msg.what = 10
msg.obj = "hello"
messenger.send(msg)
```

2. 바인딩 서비스

- 외부 앱 연동
  - 외부 앱의 서비스를 bindService() 함수로 실행하려면 먼저 서비스를 등록한 매니페스트에 외부 앱을 연동할 수 있게끔 <intentfilter>가 선언
  - bindService() 함수로 실행하는 앱에서는 외부 앱에 접근할 수 있도록 매니페스트에 queries 등록

• 매니페스트에 인텐트 필터 선언

```
<service
    android:name=".MyService"
    android:exported="true">
    <intent-filter>
        <action android:name="ACTION_OUTER_SERVICE" />
    </intent-filter>
</service>
```

• 매니페스트에 패키지 정보 등록

```
<manifest  ... 생략 ... >
    (... 생략 ...)
    <queries>
        <package android:name="com.example.test_outter" />
    </queries>
</manifest>
```

2. 바인딩 서비스

외부 앱 연동

- 외부 앱을 연동하고자 한다면 bindService() 함수로 발생하는 인텐트에 실행 대상인 앱의 패키지명을 명시
- 프로세스 간 통신에서는 주고받는 데이터는 Parcelable이나 Bundle 타입이어야 함

• 실행할 앱의 패키지명 명시

```
val intent = Intent("ACTION_OUTER_SERVICE")
intent.setPackage("com.example.test_outter")
bindService(intent, connection, Context.BIND_AUTO_CREATE)
```

• 번들 객체 이용

```
val bundle = Bundle()
bundle.putString("data1", "hello")
bundle.putInt("data2", 10)

val msg = Message()
msg.what = 10
msg.obj = bundle
messenger.send(msg)
```

2. 바인딩 서비스

AIDL 통신 기법

- AIDL은 두 프로세스 사이에 데이터를 주고받는 프로세스 간 통신
- 서비스 컴포넌트의 bindService() 함수를 이용
- 프로세스 간 통신은 앞에서 살펴본 메신저를 이용하면 AIDL보다 더 쉽게 구현
- 메신저를 이용하는 방법은 플랫폼에서 제공하는 API를 이용해야 하므로 주고받는 데이터의 종류가 많을 때는 효율이 떨어질 수 있음.
- 메신저는 모든 외부 요청을 싱글 스레드에서 처리
- AIDL은 여러 요청이 들어오면 멀티 스레드 환경에서 동시에 실행

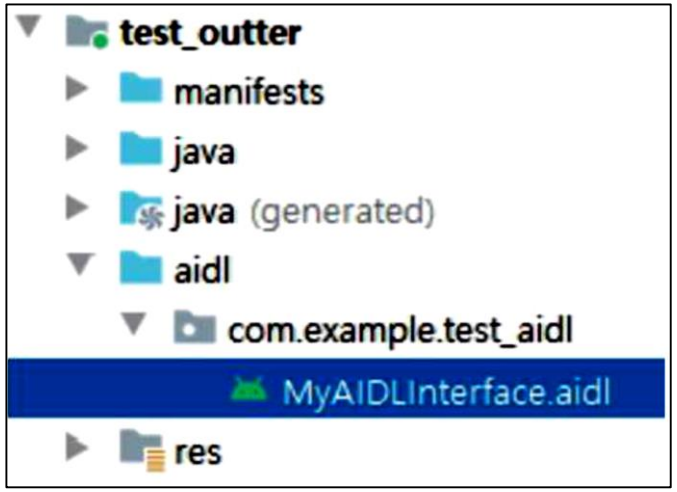
2. 바인딩 서비스

- 서비스를 제공하는 앱
  - AIDL을 이용하려면 우선 확장자가 aidl인 파일을 만들어야 함.
  - AIDL 파일에 선언된 함수를 구현해 실제 작업을 처리하는 내용을 작성하는 곳은 서비스
  - AIDL은 바인드 서비스를 이용하므로 onBind() 함수에서 서비스를 인텐트로 실행한 곳에 객체를 전달
  - 이때 AIDL 파일을 구현한 객체가 아니라 프로세스 간 통신을 대행해 주는 Stub를 전달

• AIDL 파일 내용

```
package com.example.test_aidl;

interface MyAIDLInterface {
    void funA(String data);
    int funB();
}
```



• 서비스 컴포넌트 구현

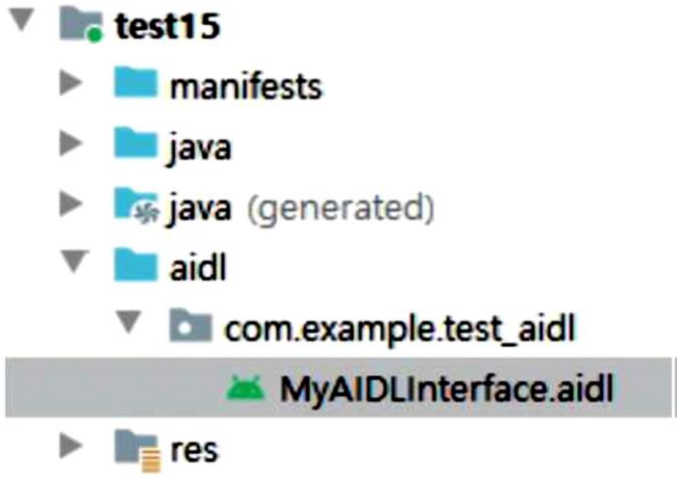
```
class MyAIDLService : Service() {
    override fun onBind(intent: Intent): IBinder {
        return object : MyAIDLInterface.Stub() {
            override fun funA(data: String?) {
            }
            override fun funB(): Int {
                return 10
            }
        }
    }
}
```

2. 바인딩 서비스

- 서비스를 이용하는 외부 앱
  - bindService() 함수를 이용하며 인텐트에 패키지 정보를 포함

• 매니페스트에 패키지 정보 등록

```
<queries>
    <package android:name="com.example.test_outter" />
</queries>
```



  - AIDL 서비스를 이용하는 앱도 AIDL 서비스를 제공하는 앱에서 만든 AIDL 파일을 가지고 있어야 함.

2. 바인딩 서비스

• 외부 앱의 서비스 실행

```
class MainActivity : AppCompatActivity() {
    lateinit var aidlService: MyAIDLInterface
    (... 생략 ...)
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        (... 생략 ...)
        val intent = Intent("ACTION_AIDL_SERVICE")
        intent.setPackage("com.example.test_outter")
        bindService(intent, connection, Context.BIND_AUTO_CREATE)
    }
    (... 생략 ...)
    val connection: ServiceConnection = object : ServiceConnection {
        override fun onServiceConnected(name: ComponentName?, service: IBinder?) {
            aidlService=MyAIDLInterface.Stub.asInterface(service)
        }
       override fun onServiceDisconnected(name: ComponentName?) {
           Log.d("kkang", "onServiceDisconnected...")
       }
   }
}
```

3. 백그라운드 제약

리시버의 백그라운드 제약

- 브로드캐스트 리시버는 암시적 인텐트로 실행할 수 없음.
- 매니페스트에 등록한 리시버를 명시적으로 실행하는 것은 정상

```
Background execution not allowed: receiving Intent { act=ACTION_RECEIVER flg=0x10 } to
com.example.test15/.MyReceiver
```

- registerReceiver() 함수로 등록하면 암시적 인텐트로도 잘 실행

• 코드에서 브로드캐스트 리시버 등록

```
receiver = object : BroadcastReceiver() {
    override fun onReceive(context: Context?, intent: Intent?) {
        Log.d("kkang", "outer app dynamic receiver")
    }
}
registerReceiver(receiver, IntentFilter("ACTION_OUTER_DYNAMIC_RECEIVER"))
```

• 암시적 인텐트로 브로드캐스트 리시버 실행

```
val intent = Intent("ACTION_OUTER_DYNAMIC_RECEIVER")
sendBroadcast(intent)
```

3. 백그라운드 제약

▪ 서비스의 백그라운드 제약

- 서비스는 앱이 백그라운드 상태일 때 인텐트를 전달하면 오류가 발생

```
Not allowed to start service Intent { act=ACTION_OUTER_SERVICE pkg=com.example.test_
outter }: app is in background uid null
```

- 포그라운드 상황
  - 액티비티가 시작되든 일시 중지되든 상관없이 보이는 액티비티가 있을 때
  - 포그라운드 서비스가 있을 때
  - 앱의 서비스에 바인딩하거나 앱의 콘텐츠 프로바이더를 사용해 또 다른 포그라운드 앱이 연결되었을 때

3. 백그라운드 제약

▪ 서비스의 백그라운드 제약

- 앱이 백그라운드 상황이더라도 다음과 같은 경우에는 서비스가 정상으로 실행
  - 우선순위가 높은 파이어베이스 클라우드 메시징(FCM) 처리
  - SMS/MMS 메시지와 같은 브로드캐스트 수신
  - 알림에서 PendingIntent 실행
  - VPN 앱이 포그라운드로 승격되기 전에 VpnService 시작
- startForegroundService() 함수로 인텐트를 시작하면 앱이 백그라운드 상황에서도 서비스가 실행

• 안드로이드 버전에 따라 백그라운드 상황 대처

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
    startForegroundService(intent)
} else {
    startService(intent)
}
```

3. 백그라운드 제약

▪ 서비스의 백그라운드 제약

- 앱이 백그라운드 상황에서 startForegroundService() 함수로 실행한 서비스는 얼마 후 다음과 같은 오류가 발생하면서 강제로 종료

```
Context.startForegroundService() did not then call Service.startForeground()
```

- startForegroundService() 함수로 실행했다면 빨리 startForeground() 함수를 호출해 포그라운드 상황으로 만들라는 의미

• 서비스 쪽 코드

```
val notification = builder.build()
startForeground(1, notification)
```

• 매니페스트에 퍼미션 등록

```
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
```

4. 잡 스케줄러

- 잡 스케줄러의 실행조건
  - 잡 스케줄러에 조건으로 명시할 수 있는 상황
  - 네트워크 타입
  - 배터리 충전 상태
  - 특정 앱의 콘텐츠 프로바이더 갱신(대표적으로 갤러리 앱)
  - 실행 주기
  - 최소 지연 시간
  - 시스템 재구동 시 현재 조건 유지 여부

4. 잡 스케줄러

잡 스케줄러의 3가지 구성 요소

- 잡 스케줄러는 다음 3가지 요소로 구성
  - 잡 서비스: 백그라운드에서 처리할 작업을 구현한 서비스
  - 잡 인포: 잡 서비스 정보와 실행될 조건을 지정
  - 잡 스케줄러: 잡 인포를 시스템에 등록

4. 잡 스케줄러

- 잡 서비스 — 백그라운드 작업 구현
  - 잡 서비스는 개발자가 만드는 서비스이므로 매니페스트에 <service> 태그로 등록
  - android.permission.BIND_JOB_SER VICE 퍼미션도 포함

• 매니페스트에 잡 서비스 등록

```
<service
    android:name=".MyJobService"
    android:enabled="true"
    android:exported="true"
    android:permission="android.permission.BIND_JOB_SERVICE"></service>
```

4. 잡 스케줄러

- 잡 서비스 작성
  - JobService를 상속받아 작성
  - onStartJob(), onStopJob() 함수는 반드시 재정의
  - onStartJob() 함수에는 백그라운드에서 처리할 작업을 구현
  - 함수의 반환값은 Boolean 타입인데 true인지 false인지에 따라서 다르게 동작
    - false: 작업이 완벽하게 종료되었음을 의미
    - true: 작업이 아직 끝나지 않았음을 의미
  - onStopJob() 함수의 반환값도 Boolean 타입인데 true, false에 따라 다르게 동작
    - false: 잡 스케줄러 등록을 취소
    - true: 잡 스케줄러를 재등록

• 잡 서비스 클래스 작성

```
@TargetApi(Build.VERSION_CODES.LOLLIPOP)
class MyJobService : JobService() {
    override fun onCreate() {
        super.onCreate()
        Log.d("kkang","MyJobService.......onCreate()")
    }
    override fun onDestroy() {
        super.onDestroy()
        Log.d("kkang","MyJobService.......onDestroy()")
    }
    override fun onStartJob(params: JobParameters?): Boolean {
        Log.d("kkang","MyJobService.......onStartJob()")
        return false
    }
    override fun onStopJob(params: JobParameters?): Boolean {
        Log.d("kkang","MyJobService.......onStopJob()")
        return false
    }
}
```

4. 잡 스케줄러

- 잡 인포 — 잡 서비스의 실행 조건 정의
  - JobInfo.Builder에 지정한 잡 서비스가 실행되는 조건을 명시
    - setPersisted(true): 기기를 재부팅해도 작업 등록을 유지해야 하는지를 설정
    - setPeriodic(long intervalMillis): 작업의 실행 주기를 설정
    - setMinimumLatency(long minLatencyMillis): 작업의 실행 지연 시간을 설정
    - setOverrideDeadline(long maxExecutionDelayMillis): 다른 조건에 만족하지 않더라도 작업이 이 시간 안에 실행되어야 함을 설정
    - setRequiredNetworkType(int networkType): 네트워크 타입을 설정
    - setRequiresBatteryNotLow(Boolean batteryNotLow): 배터리가 낮은 상태가 아님을 설정
    - setRequiresCharging(boolean requiresCharging): 배터리가 충전 상태인지를 설정

• 잡 서비스의 실행 조건 정의

```
var jobScheduler: JobScheduler? = getSystemService<JobScheduler>()

JobInfo.Builder(1, ComponentName(this, MyJobService::class.java)).run {
    setRequiredNetworkType(JobInfo.NETWORK_TYPE_UNMETERED)
    jobScheduler?.schedule(build())
}
```

4. 잡 스케줄러

- 잡 스케줄러 — 잡 서비스 등록 시 데이터 전달
  - 잡 서비스를 JobInfo 객체를 이용해 시스템에 등록하면 조건에 만족할 때 실행
  - 잡 서비스에 데이터를 전달하려면 JobInfo.Builder의 setExtras() 함수를 이용
  - 전달한 데이터를 잡 서비스에서 가져올 때는 onStartJob() 함수의 매개변수를 이용() 함수를 이용

• 잡 서비스에 데이터 전달

```
var jobScheduler: JobScheduler? = getSystemService<JobScheduler>()
val extras = PersistableBundle()
extras.putString("extra_data", "hello kkang")
val builder = JobInfo.Builder(1, componentName)
builder.setRequiredNetworkType(JobInfo.NETWORK_TYPE_UNMETERED)
builder.setRequiresCharging(true)
builder.setExtras(extras)
val jobInfo = builder.build()
jobScheduler!!.schedule(jobInfo)
```

• 잡 서비스에서 데이터 가져오기

```
override fun onStartJob(jobParameters: JobParameters): Boolean {
    jobParameters.extras.getString("extra_data")
    (... 생략 ...)
    return false
}
```

실습:MP3 재생 앱 만들기

- 1단계. 모듈 생성하기
  - Ch15_Service와 Ch15_Outer라는 이름의 모듈을 만든다.
  - Ch15_Outer는 음악을 재생하는 앱
  - Ch15_Service는 프로세스간 통신으로 Ch15_Outer 앱을 조종하는 리모컨같은 역할
- 2단계. 빌드 그래들 작성하기
  - 뷰 바인딩을 이용하는 설정과 코루틴을 사용하는 라이브러리를 등록
- 3단계. 실습 파일 복사하기
  - res 디렉터리의 drawable, layout 디렉터리를 Ch15_Service 모듈의 같은 위치에 복사
  - 코틀린 파일이 있는 디렉터리에서 MainActivity.kt 파일을 복사해 Ch15_Service 모듈의 소스 영역에 덮어쓰기

실습 : MP3 재생 앱 만들기

- 4단계. 테마 파일 수정하기
  - 액션바가 출력되지 않게 수정
- 5단계. 재생할 음원 파일 준비하기
  - res/raw 디렉터리를Ch15_Outter 모듈의res 위치에 복사
- 6단계. 메신저 서비스 작성하기
  - [New → Ser vice → Service]
  - 클래스 이름에 My MessengerService를 입력
- 7단계. AIDL 서비스 작성하기
  - [New → Folder → AIDL Folder]
  - [New → AIDL → AIDL File]
  - 이름에 MyAIDL Interface라고 입력

실습 : MP3 재생 앱 만들기

- 8단계. 매니페스트에 패키지 공개 등록하기
  - 외부 모듈의 패키지에 접근할 수 있도록 추가
- 9단계. 메인 액티비티에서 메신저 이용 코드 작성하기
  - MainActivity.kt 파일을 열고 Ch15_Outer에서 제공하는 메신저를 프로세스 간 통신으로 이용하는 코드를 작성
- 10단계. 메인 액티비티에서 AIDL 이용 코드 작성하기
  - Ch15_Service 모듈의MainActivity.kt 파일을 열고 Ch15_Outer에서 제공하는 AIDL을 프로세스간 통신으로 이용하는 코드를 추가
- 11단계. 메인 액티비티에서 잡 스케줄러 이용 코드 작성하기
  - Ch15_Service 모듈에 MyJobService라는 이름으로 서비스를 만들고 작성

실습 : MP3 재생 앱 만들기

- 12단계: 앱 실행