● 서비스: 백그라운드 작업을 위한 컴포넌트로 화면과 상관없이 장시간 동안 처리해야 하는 업무를 구현할 때 사용한다. 하지만 시간이 오래 걸리는 업무라도 앱의 화면이 스마트폰에 떠 있는 동안만 진행되어야 한다면, 서비스가 아닌 액티비티로 작성하는 게 좋다. 즉, 서비스는 다른 앱이 사용자 화면을 점유하고 있더라도 앱의 프로세스가 살아서 계속 무언가 작업을 수행해야 할 때 사용하는 컴포넌트이다.

▶ 작성 방법: Service라는 클래스를 상속 받아서 작성한다.

```
public class PlayService extends Service {
    //...
}
```

이렇게 작성한 서비스도 컴포넌트 클래스이므로 AndroidManifest.xml 파일에 등록해야 한다. 액티비티나 브로드캐스트 리시버와 마찬가지로 암시적 인텐트로 수행되게 하려면 <intent-filter>를 서브 태그로 등록해 주어야 한다.

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

▶ 실행/종료 방법: 서비스를 실행하고 종료하기 위한 두 가지 함수가 있다. 서비스를 실행하면서 인텐트의 Extra 데이터를 전달할 수 있다. 그런데 서비스는 액티비티나 브로드캐스트 리시버와는 다르게 서비스를 종료하는 인텐트 발생도 제공한다. (서비스를 구동시킨 곳에서 종료 인텐트를 날리는 것이 아니다)

1. startService(), stopService()

```
Intent intent = new Intent(this, PlayService.class);
startService(intent);
```

```
Intent intent = new Intent(this, PlayService.class);
stopService(intent);
```

2. bindService(), unbindService(): 서비스를 bindService() 함수로 실행하려면, ServiceConnection 인터페이스를 구현한 객체를 준비해야 한다. ServiceConnection 인터페이스의 추상 함수인 onServiceConnected()는 bindService() 함수로 인해 서비스가 구동하는 시점에 호출되며, onServiceDisconnected() 함수는 unbindService() 함수로 인해 서비스가 종료된 시점에 호출된다.

```
ServiceConnection connection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName, IBinder service) {
    }

    @Override
    public void onServiceDisconnected(ComponentName name) {
    }
};
```

bindService() 함수로 서비스를 구동하면서 두 번째 매개변수로 ServiceConnection을 구현한 객체를 알려주어야 한다. bindService() 함수로 실행된 서비스는 unbindService() 함수로 종료할 수 있다.

```
Intent intent = new Intent(this, BindService.class);
bindService(bIntent, connection, Context.BIND_AUTO_CREATE);
// ....
unbindService(connection);
```

▶ 서비스 생명주기: 서비스는 구동 방법이 startService()와 bindService() 두 가지이므로 어느 함수로 실행하는지에 따라 생명주기가 다르다.

1. startService() 함수로 서비스를 실행하고, stopService() 함수로 종료할 때의 서비스 생명주기

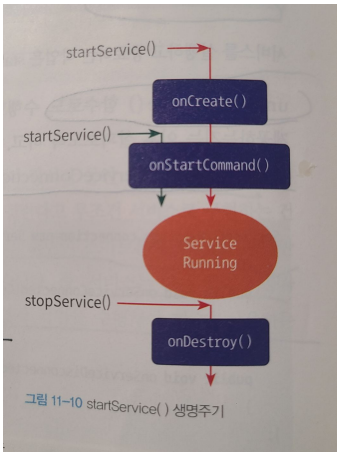

그림 11-10 startService() 생명주기

startService() 함수가 서비스를 실행하면 onCreate() → onStartCommand() 함수가 자동으로 호출되면서 Service Running 상태가 되고, stopService() 함수가 서비스를 종료할 때는 onDestory() 함수가 자동으로 호출되면서 종료된다.

서비스는 액티비티와 다르게 싱글톤으로 동작하는 컴포넌트다. 이미 실행되어 Running 상태인 서비스를 다시 startService() 함수로 실행되면 객체는 다시 생성되지 않는다. onCreate() 함수도 호출되지 않고, onStartCommand() 함수는 다시 호출된다. 즉, onCreate()는 최초 한 번만 호출되는 함수이고, onStartCommand()는 반복 호출할 수 있는 함수이다.

2. bindService() 함수로 서비스를 실행하고, unbindService() 함수로 종료할 때의 서비스 생명주기

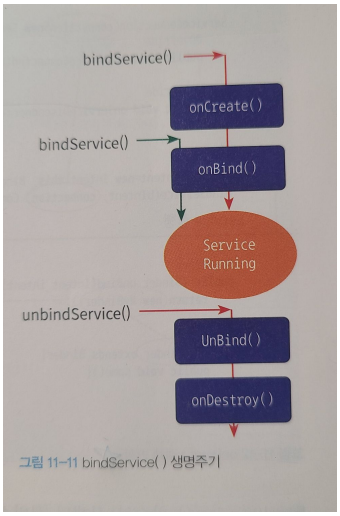

그림 11-11 bindService() 생명주기

bindService() 함수가 서비스를 시작하면 onCreate() → onBind() 함수가 호출되면서 Running 상태가 되고, unbindService() 함수로 서비스를 종료하면서 unBind() → onDestroy() 함수가 호출된다. 서비스는 싱글톤이며 이미 구성된 서비스를 다시 bindService()로 실행하면 객체가 매번 생성되지 않고 onBind() 함수만 반복 호출된다.
bindService() 함수로 인해 호출되는 onBind() 함수는 다른 생명주기 함수와 다르게 반환값이 있다.

onBind() 함수에서 반환할 객체는 Binder을 상속받아 개발자가 만든 클래스의 객체이다. Binder을 상속받아야 한다는 규칙 외에 클래스 내부의 함수 구현 규칙은 없고, 개발자가 임의로 함수를 정의하면 된다. 이 클래스의 객체를 생성해서 onBind() 함수에서 반환하면 서비스가 bindService() 함수로 실행한 곳에 전달된다. 함수 이름처럼 서비스가 실행되면서 onBind() 함수에서 반환한 객체가 서비스를 실행한 곳에 바

인딩 되는 방법인 것이다. onBind() 함수에서 반환한 객체는 bindService() 함수의 매개변수로 지정한 ServiceConnection 구현 객체의 onServiceConnected() 함수의 매개변수로 전달된다. bindService() 함수로 서비스를 실행하면 IBinder을 구현한 객체가 바인딩 된다.

```
@Override
public IBinder onBind(Intent intent) {
    return new MyBinder();
}

class MyBinder extends Binder {
    public void some() {
        // ...
    }
}
```

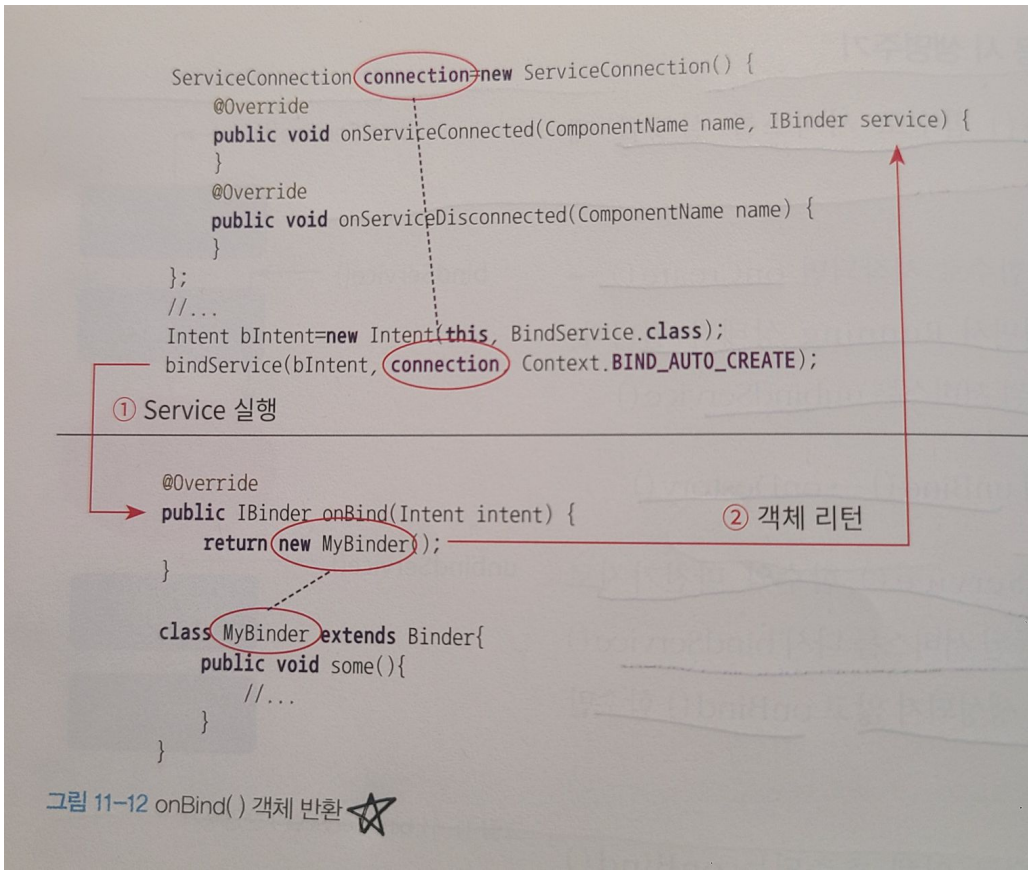

그림 11-12 onBind( ) 객체 반환

bindService() 함수로 실행된 서비스는 자신에게 바인드된 컴포넌트가 하나도 없으면 자동으로 종료된다. 즉, 하나의 액티비티가 bindService() 함수로 서비스를 실행한 것이라면, 해당 액티비티가 종료되는 시점이나, unBindService() 함수로 종료를 명시하는 시점에 서비스도 함께 종료된다. 물론 여러 개의 액티비티가 bindService() 함수로 서비스를 이용하고 있다면, 하나의 액티비티가 종료되거나 unBindService() 함수를 호출하더라도 종료되지 않고 계속 움직이게 된다.

▶ Messenger 바인딩: bindService() 함수로 서비스를 실행하면 onBind() 함수에서 서비스로 구동시킨 쪽에 객체를 넘겨주어야 하는데, 이때 이 객체를 개발자가 직접 준비한 클래스의 객체가 아니라, Messenger 클래스를 이용해서 Messenger 객체를 반환시켜 바인딩하는 방법도 있다.

우선 Service에 Handler을 상속받은 클래스를 준비한다. 서비스를 실행한 곳(외부, 액티비티)에서 데이터를 넘기면, 이 클래스의 handleMessage() 함수가 자동 콜돼서 데이터를 받아서 처리한다.

```
public class PlayService extends Service {

    class MessengerHandler extends Handler {
        @Override
        // Message msg가 외부에서 넘긴 데이터
        public void handleMessage(@NonNull Message msg) {
            // 만약 1번 데이터이면
            if (msg.what == 1) {
                // 실제 데이터(Object 타입)을 필요한 데이터 타입으로 캐스팅해서 띄우기
                Toast.makeText(getApplicationContext(), "what==1 : " + (String)msg.obj,
            } else if (msg.what == 2) {
                Toast.makeText(getApplicationContext(), "what==2 : " + (String)msg.objt,
            } else {
                super.handleMessage(msg)
            }
        }
    }

    @Override
    public IBinder onBind(Intent intent) {
        Messenger messenger = new Messenger(new MessageHandler());
        // 메신저의 생성자로 데이터를 받아 처리할 Handler를 구현한 객체를 넘겨주기
        return messenger.getBinder(); // 서비스를 실행시킨 곳에 Messenger 객체 바인딩(주기)
    }

}
```

서비스를 실행한 곳에서 데이터를 전달하면, handleMessage() 함수가 자동으로 호출된다. 이때 외부에서 전달한 데이터는 Message 타입이며, Message의 int 타입인 what 변수로 어떤 성격의 데이터인지를 구분하고 (1번 데이터인지, 2번 데이터인지...) Object 타입의 obj 변수로 전달된 데이터(실제 데이터)를 획득한다.

Messenger을 이용하기 위해, onBind() 함수의 반환값으로 Messenger 객체를 생성하면서 생성자 매개변수로 Handler을 구현한 객체를 지정한다. 그리고 Messenger 객체를 onBind() 함수의 결과값으로 반환한다. 그러면 서비스를 실행한 곳에 Messenger 객체가 전달된다.

```
public class MainActivity extends AppcompatActivity {
    Messenger messenger;

    ServiceConnection connection = new ServiceConnection() {
        // IBinder 객체는 서비스에서 넘어온 객체(onBind()가 리턴한 Messenger 객체)
        @Override
        public void onServiceConnected(ComponentName componentName, IBinder service) {
            // 서비스에서 넘어온 객체를 매개변수로 메신저 생성, 초기화
            messenger = new Messenger(iBinder);
        }

        @Override
        public void onServiceDisconnected(ComponentName componentName) {

        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceStates);
        ActivityMainBinding binding = ActivityMainBinding.inflate(getLayoutInflator());
        setContentView(binding.getRoot());
        // ...
        Intent intent = new Intent(this, PlayService.class);
        bindService(intent, connection, Context.BIND_AUTO_CREATE);

        binding.button.SetOnClickListener(view -> {
            try {
                Message message = new Message();
                message.waht = 1;
                message.obj = "hello";
                messenger.send(message);
            } catch (Exception e) {
                e.printStackTrace();
            }
        });
    }
}
```

서비스를 이용하는 액티비티 코드다. bindService() 함수로 서비스를 실행하고, 서비스에서 넘어온 객체는 onServiceConnected() 함수의 매개변수로 받는다. 이렇게 서비스로부터 넘어온 객체를 Messenger의 생성자 매개변수에 지정만 해주면 된다. 그리고 이렇게 서비스를 실행한 후 서비스에 데이터를 전달하고 싶을 때는 Messenger의 send() 함수를 호출한다. send() 함수의 매개변수는 전달하는 데이터를 추상화한 Message 객체이다. 액티비티에서 send() 함수를 호출하는 순간, 서비스의 handleMessage() 함수가 자동으로 호출된다.

- MainActivity: bindService() 함수로 서비스 실행(구동)
- Service: onBind() 함수로 인해 Messenger 객체(IBinder) 반환 및 MainActivity에 전달
- MainActivity: onServiceConnected() 함수의 매개변수로 서비스에서 넘어온 객체 받기. 그 객체(IBinder)를 이용해서 Messenger 객체 생성 및 초기화.
- MainActivity: messenger.send(message)로 서비스에 데이터 전달.
- Service: Handler을 구현한 클래스의 handleMessage() 함수 자동 Call. 받은 데이터를 처리. onBind() 함수에서 Messenger 객체를 생성하면서 생성자 매개변수로 Handler를 구현한 객체를 지정하고, Messenger 객체(IBinder)를 결과값으로 반환하여 서비스를 실행한 MainActivity에 Messenger 객체 전달.
- MainAcitivty: onServiceConnected() 함수의 매개변수로 서비스에서 넘어온 Messenger 객체 받기. 그 객체(IBinder)를 Messenger의 생성자 매개변수에 지정하여 Messenger 객체 생성 및 초기화.

▶ 패키지 공개 상태: 앱에서 외부 앱의 패키지 정보에 접근하려면 AndroidManifest.mxl 파일에 접근하고자 하는 외부 앱의 패키지명을 명시적으로 선언해야 한다.

외부 앱의 service를 앱의 액티비티에서 bindService()로 연동할 경우(이용할 경우), 패키지 정보를 지정해 주어야 한다. 외부 앱을 연동하려면 패키지 정보에 접근하기 위해 패키지 공개 상태(package visibility)라는 개념이 필요하다. 패키지 공개 상태란 AndroidManifest.xml 파일에 접근하고자 하는 외부 앱의 패키지 정보를 명시적으로 선언하는 것을 의미한다. 대표적으로 다음의 함수를 이용해서 패키지 정보에 접근하고, 이 경우 AndroidManifest.xml 파일에 외부 앱의 패키지명을 등록(패키지 공개 상태를 설정)하는 것이 필요하다.

☞ PackageManager.getPackageInfo()
☞ PackageManager.queryIntentActivities()
☞ Intent.resolveActivity()
☞ PackageManager.getInstalledPackageds()
☞ PackageManager.getInstalledApplications()
☞ bindService()

```
// AndroidManifest.xml

<manifest  >
    <queries>
        <package android:name="com.example.test4_outter" />
    <queries>
</manifest>
```

<queries> 태그 하위에 <package> 태그를 이용하여 접근하고자 하는 앱의 패키지명을 명시적으로 선언한 후 사용한다. 하나 이상일 경우 <pacakge> 태그로 계속해서 추가하면 된다. 만약 여러 앱의 패키지 정보에 접근해야 한다면, 위의 코드처럼 작성하지 않고 모든 외부 앱의 패키지 정보에 접근 허용을 해달라는 퍼미션을 선언해도 되지만, 권장사항은 <package> 태그를 이용하는 방식이다.

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

패키지 공개 상태 설정 없이 다음과 같은 코드를 실행하면 안드로이드 버전 11 이상부터는 android.content.pm.PackageManager$NameNotFoundException: com.example.test4_outter 에러가 발생한다.

```
try {
    PackageInfo packageInfo = getPackageManager().getPackageInfo("com.exmple.test4_outte
    String versionName = packageInfo.versionName;
    Log.d("kkang", versionName);
} catch (Exception e) {
    e.printStackTrace();
}
```