● 퍼미션(permission): AndroidManifest.xml의 들어가는 설정이다. 안드로이드는 컴포넌트를 이용한 앱과 앱 사이의 연동이 빈번한데, 이러한 연동에서 어떤 앱이 permission을 부여했다면, 그 앱을 이용하는 앱은 uses-permission을 선언해야 한다. permission은 자신의 앱이 외부에서 이용될 때 권한을 부여하여, 해당 권한을 가지고 들어올 때만 실행되게 하기 위한 설정이다. 앱의 컴포넌트를 보호하고 싶을 때 permission을 선언하고, 그렇게 선언된 앱을 이용하려면 uses-permission을 선언하는 구조이다.

```
// AndroidManifest.xml

<permission android:name="com.test.permission.SOME_PERMISSION"
    android:label="SOME Permission"
    android:description="@string/permission"
    android:protectionLevel="normal" />
```

name의 퍼미션의 이름이고, label과 description은 사용자에 대한 문자열로 앱의 권한 인증 화면에 출력되는 퍼미션에 대한 설명이다. 또한, protectionLevel로 보호 수준을 명시할 수 있다. normal은 낮은 수준의 보호(퍼미션으로 보호되는 기능이 사용자에게 위험 부담이 적음, 사용자에게 권한 부여 요청이 필요 없는 경우, uses-permission은 요구하지만 사용자에게 퍼미션 부여를 요구하지는 않음), dangerous는 높은 수준의 보호(퍼미션으로 보호되는 기능이 사용자에게 위험 부담이 크다는 의미, 사용자에게 권한 부여 요청이 필요한 경우, uses-permission도 요구하고 사용자에게 퍼미션 부여도 요구한다), signature (동일한 키로 서명된 앱만 실행), signatureOrSystem(안드로이드 시스템 앱이거나 외부 앱이 동일 키로 서명되어 있어야 실행)가 있다. 이렇게 AndroidManifest.xml에 permission 태그를 추가한 다음에는, 보호하고 싶은 컴포넌트에 퍼미션을 적용해야 한다.

```
<activity android:name=".SomeActivity"
  android:permission="com.test.permission.SOME_PERMISSION">
  <intent-filter>
    <action adroid:name="AAA"/>
    <category android:name="android.intent.category.DEFAULT"/>
  </intent-filter>
</activity>
```

이렇게 보호된 컴포넌트를 이용하는 앱은 AndoridManifest.xml에 uses-permission이 등록되어 있어야 한다.

```
<manifest xmlns:android="https://schemas.android.com/apk/res/android"
  package="com.example.test">

  <uses-permission android:name="come.test.permission.Some_PERMISSION"/>
  <!--중략-->
</manifest>
```

▶ 시스템의 퍼미션: 다른 앱을 연동하는 것이 아니지만 특정 기능은 시스템에서 보호하고 있기 때문에 앱에서 그 기능을 이용할 때 uses-permission을 선언하지 않으면 에러가 발생한다. 대표적인 예가 진동 알람을 위한 퍼미션이다. 앱에서 진동 알림을 울릴 때 외부 앱과 연동하진 않지만, 해당 기능을 시스템 자체에서 퍼미션으로 보호하고 있어서 uses-permission을 선언해 주어야 한다. 안드로이드에서는 퍼미션을 개별로 나누어 관리하는 게 아니라, 관련된

퍼미션을 묶어서 관리하고 사용자에게 알려줄 때는 그둡 단위로 알려준다. 예를 들어, 위치를 추적하기 위한 퍼미션은 ACCESS_COARSE_LOCAGTION과 ACESS_FINE_LOCATION으로 구분되어 있다. 하지만 사용자 권한 화면에서는 2가지의 퍼미션을 따로 보여주지 않고, LOCATION 그둡으로 묶어 하나로 알려준다.

퍼미션 그둡	퍼미션
CALENDAR	READ_CALENDAR
	WRITE_CALENDAR
CAMERA	CAMERA
CONTACTS	READ_CONTACTS
	WRITE_CONTACTS
	GET_ACCOUNTS
LOCATION	**ACCESS_FINE_LOCATION**: 정확한 위치 정보 액세스
	ACCESS_COARSE_LOCATION
MICROPHONE	RECORD_AUDIO
PHONE	**READ_PHONE_STATE**: 장치의 전화번호, 네트워크 정보, 진행 중인 통화 상태 등 전화 상태에 대한 읽기
	CALL_PHONE
	READ_CALL_LOG
	WRITE_CALL_LOG
	ADD_VOICEMAIL
	USE_SIP
	PROCESS_OUTGOING_CALLS
SENSORS	BODY_SENSORS
SMS	**SEND_SMS**: SMS 메시지 발신
	RECEIVE_SMS: SMS 메시지 수신
	READ_SMS: SMS 메시지 읽기
	RECEIVE_WAP_PUSH
	RECEIVE_MMS
STORAGE	READ_EXTERNAL_STORAGE
	WRITE_EXTERNAL_STORAGE: 외부 저장소에 파일 쓰기
	ACCESS_NETWORK_STATE: 네트워크에 대한 정보 엑세스
	ACCESS_WIFI_STATE: 와이파이 네트워크에 대한 정

	보 엑세스
	BATTERY_STATS: 배터리 통계 수집
	BLUETOOTH: 연결된 블루투스 장치에 연결
	BLUETOOTH_ADMIN: 블루투스 장치를 검색하고 페어링
	CAMERA: 카메라 장치에 액세스
	INTERNET: 네트워크 연결
	READ_EXTERNAL_STORAGE: 외부 저장소에서 파일 읽기
	RECEIVE_BOOT_COMPLETED: 부팅 완료 시 수행
	VIBRATE: 진동 울리기

퍼미션의 protectionLevel에 따라 스마트폰의 환경설정 권한 화면에 출력되는 퍼미션이 다르다. 예를 들면 AndroidManifest.xml에 다음처럼 uses-permission이 부여된 앱이 있다고 가정하자.

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

위의 퍼미션은 모두 시스템에서 부여한 것이다. android.permission.ACCESS_NETWORK_STATE은 스마트폰의 네트워크 정보에 접근할 때 필요한 퍼미션으로, protectionLevel이 "normal"로 선언되어 있다. android.permission.ACCESS_FINE_LOCATION은 사용자 위치를 추적할 때 필요한 퍼미션으로, protectionLevel이 "dangerous"로 선언되어 있다. 이 uses-permission이 부여된 앱을 스마트폰의 환경설정 권한 화면에서 확인해 보면, normal로 선언된 퍼미션은 권한 화면에 출력되지 않는 반면, dangerous로 선언된 퍼미션은 권한 설정 화면에서 사용자에게 권한을 부여할지를 물어본다. 사용자에게 위험이 큰 퍼미션이기 때문이다.

▶ 퍼미션 허용: 안드로이드 6.0부터는 사용자가 퍼미션을 거부할 수 있게 변경되었다. 사용자가 퍼미션을 거부해버리면 AndroidManifest.xml 파일에 uses-permission 선언이 안 된 것처럼 되어버려서, 앱 설치 후 사용자가 퍼미션을 거부하는 상황을 고려해서 프로그램을 작성해야 한다. 가장 먼저 현시점에 퍼미션 상태를 확인하는 코드가 필요하다. checkSelfPermission() 함수를 이용하면, 퍼미션의 상태를 확인할 수 있다.

```
int checkSelfPermission(Context context, String permission)
// 두 번째 매개변수가 퍼미션의 이름이다.
// 함수의 반환값으로는 PERMISSION_GRANTED(퍼미션이 부여된 상태)와
// PERMISSION_DENIED(퍼미션이 부여되지 않은 상태)가 있다.
```

CheckSelfPermission() 함수가 PERMISSION_GRANTED 값을 반환한다면, 퍼미션이 부여된 것이므로 관련 기능을 실행하면 된다.

```
if (ContextCompat.checkSelfPermission(this, Manifest.permission.READ_EXTERNAL_STORAGE) =
    // ....
}
```

만일 퍼미션이 부여되지 않았다면 기능 수행 시 에러가 발생하기 때문에, 사용자에게 퍼미션 허용을 요청해야 한다. 그런데 사용자가 직접 환경설정의 권한 화면으로 가는 건 번거롭기 때문에, 앱 안에서 퍼미션 허용 여부를 묻는 대화상자(Dialog)를 뛰워 퍼미션 허용을 요청해 주는 것이 좋다. 퍼미션 허용 여부를 묻는 대화상자는 시스템 다이얼로그이고, 사용자가 이곳에서 거부 혹은 허용을 선택하게 딘다. 사용자로부터 결과를 되돌려 받을 때 사용하는 ActivityResultLauncher로 퍼미션 허용을 요청한다. 우선, ActivityResultLauncher은 registerForActivity() 함수에 의해서 만들어진다.

```
public final <I, O> ActivityResultLauncher<I> registerForActivityResult (
    @NonNull ActivityResultContract<I, O> contract,
    @NonNull AcitivityResultCallBack<O> call back)
```

registerForActivityResult() 함수는 두 개의 매개변수를 갖는다. 첫 번째 매개변수는 ActivityResultContract 타입의 객체로 어떤 요청인지를 표현하기 위해서 사용된다. 다양한 요청을 표현하는 서브 클래스가 있으며, 퍼미션을 요청할 때는 RequestPermission을 사용한다. 두번째 매개변수는 결과를 받았을 때 호출하는 콜백이다.

```
AcitivityResultLauncher<String> permissionLauncher
  = reigsterForActivityResult(new ActivityResultContracts.RequestPermission(), isGrantec
    if (isGranted) {
        Log.d("kkang", "granted...");
    } else {
        Log.d("kkang", "denied...");
    }
});
```

AcitivtyResultLauncher 객체를 생성하면, launch() 함수를 호출하여 필요한 요청을 할 수 있다.

```
permissionLauncher.launch("android.permission.ACESS_FINE_LOCATION");
```

요청한 결과가 registerForActivityResult() 함수의 두 번째 매개변수에 등록된 콜백이 호출되면서 전달된다.

Resources: 깡쌤의 안드로이드 프로그래밍