API Level 26 (Android Oreo)부터는 백그라운드에서 실행되는 앱이 스마트폰의 메모리나 배터리 등의 리소스를 과도하게 소비하는 것을 개선하기 위해 백그라운드 실행에 제한을 두고 있다. 이 때문에 API Level 26 이상의 스마트폰에서 실행될 앱이라면 백그라운드 실행 제한을 고려하여 앱을 개발해야 한다.

● 브로드캐스트 리시버 제약: 브로드캐스트 리시버의 백그라운드 제약은 암시적 인텐트에 의한 실행 제한이다. 일반적으로 BroadcastReceiver는 시스템의 특정 상황(부팅 완료, SMS 수신 등)을 파악하기 위해 많이 사용하며, 이때 시스템에서 발생시키는 인텐트에 의해 실행되므로 AndroidManifest.xml 파일에 암시적 방법으로 등록하는데, 이 부분엔 제한이 없다. 제한이 가해지는 경우는 개발자 코드에서 BroadcastReceiver을 실행하기 위해 암시적 방법으로 인텐트를 발생시키는 경우이다.

```
// AndroidManifest.xml에 이처럼 등록된 리시버가 있다고 가정하면,
<receiver
    android:name=".MyReceiver"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action android:name="com.example.ACTION_MY_RECEIVER" />
    </intent filter>
</receiver>
```

```
// 만약 앱의 코드에서 암시적 방법으로 인텐트로 실행할 경우, 에러가 발생한다.
Intent intent = new Intent("com.example.ACTION_MY_RECEIVER");
sendBroadcast(intent);

// 물론, 명시적 방법으로는 리시버가 잘 실행된다.
Intent intent = new Intent(this, MyReceiver.class)
sendBroadcast(intent);
```

API Level 26의 변경사항의 주목적은 백그라운드 실행 제한이다. 즉, 리시버가 백그라운드에서 암시적인 방법으로 실행되는 것을 제한한 것이다. 리시버가 백그라운드가 아니라면 암시적 인텐트로도 잘 실행된다. 리시버를 AndroidManifest.xml 파일에 등록(정적)하지 않고, 코드에서 registerReceiver() 함수를 이용해서 등록(동적)하면, 액티비티 같은 특정 컴포넌트가 실행되면서 리시버가 등록된다. 그러므로 리시버 혼자 백그라운드에서 실행되는 것이 아니며, 특정 컴포넌트 작업과 연관되어 있고, 이는 백그라운드 실행으로 판단되지 않는다. 따라서 registerReceiver() 함수로 등록한 리시버는 암시적 인텐트로 실행할 수 있다.

```
registerReceiver(new MyReceiver(), new IntentFilter("com.example.ACTION_REGISTER_RECEIVE
```

● 서비스 제약: 포크라운드(foreground)에 있을 때의 서비스 이용에는 제약 사항이 없다. 다음의 경우 앱의 포그라운드로 판단한다.

> 시작되거나 일시 중지되는 것과는 상관없이 보이는 액티비티가 있는 경우 (액티비티가 포커스를 가지고 있든 없

든 어쨌든 화면에 출력되고 있는 상황)
포그라운드 서비스가 있는 경우
앱의 서비스 중 하나에 바인드하거나 앱의 콘텐츠 제공자 중 하나를 사용하여 앱에 또 다른 포그라운드 앱이 연결된 경우 (외부 앱에서 바인드 서비스를 통해 서비스를 구동할 경우 서비스의 데이터가 외부 앱으로 넘어가 화면에 나올 수도 있기 때문에. 아니면 콘텐트 프로바이더를 통해 데이터를 공유하고 있을 경우).

이 외에는 앱이 백그라운드 상황에 있다고 판단하지만, 앱이 백그라운드 상황에 있어도 다음의 경우에는 서비스가 정상적으로 실행된다.

우선 순위가 높은 파이어베이스 칼두으드 메이징(FCM) 처리
SMS/MMS 메시지와 같은 브로드캐스트 수신
알림에서 PendingIntent 실행 (알림을 터치했을 때 서비스 실행)
VPN 앱이 포그라운드로 승격되기 전에 VpnService 시작

앱의 액티비티가 화면에 보이는 것과 상관없이 실행 상태이거나, 액티비티가 실행되지 않더라도 화면과 관련된 서비스라면 정상적으로 실행된다. 대표적인 예는 알림의 프로그레스바를 지속해서 증감하는 서비스이다. 이는 화면을 위한 액티비티는 아니지만, 포그라운드를 목적으로 한다. 그리고 바인드 방법으로 이용되는 서비스도 정상적으로 실행된다.

위의 상황과 상관없이 실행되는 서비스에는 인텐트에 의한 서비스 실행에 제약이 가해진다. 대표적인 사례가 스마트폰 부팅이 완료되지마자 실행되는 리시버에서 서비스를 구동하려는 경우이다. 앱이 포그라운드 상황이 아니면 서비스 구동에 제약이 걸리는 것이다. 하지만 백그라운드 상황에서도 앱의 서비스를 정상적으로 구동할 수 있는데, 대표적인 사례가 부팅 완료 시점에 앱의 서비스를 구동하는 경우이다. 부팅 완료 시점이므로 당연히 앱은 백그라운드 상황이다.

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
    context.startForegroundService(intent1);
} else {
    context.startService(intent);
}
```

부팅 완료 시점에 실행되는 리시버의 코드이며 Android Oreo이거나 상위 버전일 때— startForegroundService() 함수에 의해 서비스 인텐트를 발생시키면 서비스가 정상 실행된다. 그러나 이 경우 실행된 서비스가 빠른 시간 내에 (몇 초 이내로) 알림 등을 위한 startForeground() 함수를 호출해야 한다. 이는 알림을 위한 함수로 서비스에서 알림을 발생시키기 위해 사용된다. 즉, 서비스를 포그라운드 서비스로 만드는 기법이다. 만약 빠든 시간 내에 startForeground() 함수를 호출하지 않으면 에러 메시지가 로그캣에 발생하며 서비스가 종료된다.

```
Notification notification = builder.build();
startForeground(1, notification);
```

startForegroundService() 함수에 의해 실행된 서비스가 startForeground() 함수를 호출한 경우이다. startForeground() 메서드를 호출한다는 것은 결국 status bar에 알림을 띄운다는 의미이다. 서비스를 구동하면서 알림을 띄우면 사용자 화면에 알림이 출력되므로 앱은 포그라운드 상황이라고 볼 수 있다. 또한 API Level 28 (Android 9.0)부터는 startForeground() 메서드를 이용해 앱을 포그라운드로 실행하면서 서비스를 구동하려면, 아래처럼 퍼미션을 설정해 주어야 한다.

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

● JobScheduler: 백그라운드 상황에서는 앱이 정상적으로 실행되지 않지만, 백그라운드 상황에서 뭔가를 처리해야 할 때도 있다. 예를 들어, 앱이 실행되지 않은 상황이지만 폰이 와이파이 네트워크에 연결된 순간을 자동으로 감지하여 서버로부터 얻은 데이터로 앱의 내용을 갱신하도록 구현해야 하는 경우가 있다. 이처럼 앱이 백그라운드 상황에 있어도 작동하는 서비스를 위해 JobScheduler을 만든다. JobScheduler는 앱이 백그라운드 상황에 있어도 서비스가 시스템에서 실행될 수 있도록 한다. 하지만 모든 상황에서 서비스가 실행되는 것은 아니고, JobScheduler를 시스템에 등록할 때, JobScheduler가 실행되어야 하는 조건을 명시해야 하고, 이 조건에 해당될 때 시스템에서 서비스를 실행한다. (특정 조건을 만족했을 때 시스템이 우리 앱의 JobService 서비스를 실행)

JobScheduler에 조건으로 명시할 수 있는 조건으 아래와 같다.

네트워크 타입 (ex. 와이파이가 네트워크에 연결된 경우. 네트워크 타입에 관한 조건. 이런 상황이면 앱이 백그라운드에 있어도 서비스를 실행시킬 수 있다)
배터리 충전 상태
특정 앱의 콘텐츠 프로바이더 갱신

또는 네트워크 타입이나 배터리 충전 상태가 바뀌거나 특정 앱의 콘텐츠 프로바이더 갱신이 발생하지 않아도 아래의 조건으로 서비스가 주기적으로 실행되도록 할 수 있다. 또한 폰이 재구동 되어도 아래의 조건이 시스템에 그대로 유지되도록 할 수 있다.

실행 주기
최소 지연 시간
시스템이 재구동될 때 현재 조건 유지 여부

이런 JobScheduler의 구성 요소는 세 가지이다.
1) JobService: 백그라운드에서 처리할 업무를 구현한 개발자 서비스 클래스
2) JobInfo: JobService가 실행되어야 할 조건이 지정된 객체
3) JobScheduler: JobInfo를 시스템에 등록시키는 객체

1. JobService: 백그라운드에서 처리할 업무를 구현한 개발자 서비스 클래스. AndroidManifest.xml 파일에 <service> 태그로 등록되어 있어야 하며, android.permission.BIND_JOB_SERVICE 퍼미션이 추가되어야 한다.

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

서비스를 만들 때 JobService를 상속받아 작성한다. 또는 서비스의 라이프사이클 함수인 onCreate, onStartCommand, 그리고 onDestroy를 추가할 수 있지만, onStartCommand 함수는 JobService가 실행될 때 호출되지 않기에 의미가 없다. JobService를 작성할 때 onStartJob() 함수와 onStopJob() 함수가 중요하다.

```
public class MyJobService extends JobService {
    @Override
    public boolean onStartJob(JobParameters jobParamters) {
        return false;
    }

    @Override
    public boolean onStopJob(JobParamters jobParameters) {
        return false;
    }
}
```

JobService가 실행되면 onStartJob() 함수는 자동으로 호출된다. 이 함수에는 주로 백그라운드에서 실행될 기능을 구현한다. onStartJob() 함수의 반환값은 boolean 타입으로, 이 값에 따라 onStartJob() 함수가 다르게 동작한다. false가 반환되면 시스템은 JobService의 업무가 정상적으로 처리되어 종료되었다고 판단하여 onDestroy() 함수를 실행시킨 후에 서비스를 종료힌다. true가 반한되면 onStartJob() 함수는 실행이 끝났만 아직 업무 처리가 진행되고 있다고 판단한다. onStartJob() 함수 내에서 스레드 프로그램으로 시간이 오래 걸리는 작업을 수행하는 경우를 대표적인 예로 들 수 있다. onStartJob() 함수는 끝났지만, 아직 스레드에 의해 업무가 처리되고 있다. 이처럼 true가 반환되면 시스템에서는 서비스를 종료시키지 않는다 - onDestory() 함수를 호출하지 않는다. 이 경우 업무가 종료된 후에 코드에서 jobFinish() 함수를 이용해 명시적으로 업무 실행이 종료되었음을 알려주어야 한다.

onStopJob() 함수는 항상 호출되지는 않는다. JobService가 실행되는 도중에 JobService를 실행시키기 위한 조건이 변경되거나, 어디선가 명시적으로 cancel() 함수를 이용해 JobService를 종료했을 때 onStopJob() 함수가 호출된다. 즉, onStopJob() 함수는 JobService의 업무 처리 상황(onStartJob()에서 true을 리턴하든 false를 리턴하든)에 상관없이 호출되며, JobService가 정상적으로 구동되기 힘든 상황에서 JobService가 종료되기 전에 처리해야 할 업무를 실행시킨다. onStopJob() 함수의 반환 타입도 boolean으로, 그 값에 따라 다르게 동작한다. onStopJob() 함수가 호출되었다는 것은 JobService가 처리해야 할 업무를 정상적으로 진행하지 못하고 종료되었다는 의미이다. 이런 상황에서 다시 시스템에 조건에 맞게 등록하면, 다시 샐행되도록 할지 반환 타입으로 명시하는 것이다. false가 반환되면 시스템에 등록이 취소되며, 다시 실행되지 않는다. true가 반환되면 시스템에 재등록이 되어, JobService가 구동되도록 조건을 재설정하면 다시 실행된다.

2. JobInfo: JobService가 실행되어야 할 조건이 지정된 객체. 실행할 JobService의 클래스명과 JobService가 실행되어야 할 조건 정보를 담는 객체이다. JobInfo 객체를 위한 Builder을 생성할 때, 생성자 매개변수에 ComponentName 타입의 객체를 등록하여 실행할 JobService 클래스에 대한 정보를 전달한다. 그리고 Builder 객체의 다양한 setter 함수를 이용해 JobService가 실행될 조건을 등록한다.

```
JobInfo.Builder builder = new JobInfo.Builder(1, new ComponentName(this, MyJobService.cl
builder.setRequiredNetworkType(JobInfo.NETWORK_TYPE_UNMETERED);
JobInfo info = builder.build();
```

setPersisted(true)	재부팅될 때 작업 등록을 유지할지 설정
setPeriodic(long intervalMillis)	작업의 실행 주기 설정
setMinimumLatency(long minLatencyMillis)	작업 실행의 지연 시간 설정
setOverrideDeadline(long maxExecutionDelayMillis)	다든 조건에 만족하지 않더라도 작업이 이 시간 안에 실행되어야 함을 설정
setRequiredNetworkType(int networkType)	네트워크 타입 설정
setRequiresBatteryNotLow(boolean	배터리가 낮은 상태가 아님을 설정

batteryNotLow)	
setRequiresCharging(boolean requiresCharging)	배터리가 충전 상태인지 설정

3. JobScheduler: JobInfo를 시스템에 등록시키는 객체. JobScheduler의 shedule() 함수를 이용해 JobInfo 객체를 시스템에 등록한다.

```
JobScheduler scheduler = (JobScheduler)getSystemService(JOB_SCHEDULER_SERVICE);
scheduler.schedule(info);
```

Resources: 깡쌤의 안드로이드 프로그래밍