# 9 백그라운드에서 작업하기

**이번 장에서 다루는 내용**

- ☑ 서비스를 만들고, 시작하고, 중지하는 법
- ☑ 서비스와 액티비티를 바인드하는 법
- ☑ 서비스의 우선순위를 포그라운드로 설정하는 법
- ☑ AsyncTask를 이용해 백그라운드 처리를 관리하는 법
- ☑ 백그라운드 스레드를 만드는 법과 핸들러를 이용해 GUI 스레드와 동기화하는 법
- ☑ 토스트를 표시하는 법
- ☑ 알림 매니저를 이용해 사용자에게 애플리케이션 이벤트를 알리는 법
- ☑ 강조 알림과 진행 중 알림을 만드는 법
- ☑ 알람을 이용해 애플리케이션 이벤트를 스케줄링 하는 법

안드로이드는 Service라는 클래스를 제공한다. 이 클래스를 이용하면 사용자 인터페이스 없이 화면에 보이지 않은 채로 실행되어야 하는 작업과 기능을 처리하기 위한 용도로 특화된 애플리케이션 컴포넌트를 만들 수 있다.

안드로이드는 서비스에게 비활성 액티비티보다 높은 우선순위를 부여하므로, 시스템이 리소스를 필요로 한다고 해서 서비스가 종료될 가능성은 적다. 실제로 런타임이 실행 중인 서비스를 시급히 종료시켜야 하는 경우, 해당 서비스는 리소스가 충분해지는 즉시 재시작되도록 구성될 수 있다. 음악이 재생 중에 끊기는 경우처럼 서비스가 죽으면 사용자 경험에 치명상을 입는 극단적인 경우도 있다. 이런 경우에는 서비스의 우선순위를 포그라운드 액티비티만큼 높일 수 있다.

서비스를 이용하면 애플리케이션이 활성화된 상태가 아닌 경우에도 계속 실행되면서 이벤트에 반응하게 할 수 있다.

서비스는 GUI 없이 동작하지만, 액티비티와 브로드캐스트 리시버처럼 애플리케이션 프로세스의 메인 스레드에서 실행된다. 이번 장에서는 애플리케이션이 좋은 반응성을 유지할 수 있도록 시간이 많이 드는 처리(네트워크 조회 같은)를 Thread와 AsyncTask 클래스를 이용해 백그라운드 스레드로 옮기는 법을 배운다.

안드로이드는 애플리케이션이 액티비티 없이도 사용자와 대화할 수 있도록 여러 가지 기법들을 제공한다. 이번 장에서는 알림과 토스트를 이용해 활성화된 애플리케이션을 방해하지 않고 사용자에게 정보를 제공하는 법을 배운다.

토스트Toasts는 일시적인 성격의 비 모들non-modal 다이얼로그 박스 메커니즘으로서, 활성화된 애플리케이션의 포커스를 빼앗지 않고 사용자에게 정보를 표시하는 데 쓰인다. 토스트를 이용하면 어떠한 애플리케이션 컴포넌트에서든 사용자의 화면 위에 간단한 메시지를 표시할 수 있다.

토스트는 조용하고 일시적인데 반해 **알림**Notifications은 사용자가 분명히 알 수 있는 방식으로 정보를 알리는 메커니즘을 제공한다. 사용자가 휴대폰을 사용하지 않을 경우에는 대부분 주머니에 넣어두거나 책상 위에 올려둔다. 벨이 울리거나, 진동이 오거나, 화면이 깜박여야 그제서야 휴대폰을 꺼내 든다. 사용자가 이러한 알림을 놓치게 되면, 상태 표시줄 아이콘이 이벤트 발생을 알리는 데 사용된다. 사용자의 주의를 끄는 이러한 모든 일들이 안드로이드에서는 알림을 통해 가능하다.

알람Alarms은 애플리케이션 수명 주기의 제어 바깥에서 정해진 시간에 인텐트를 발생시키기 위한 메커니즘을 제공한다. 알람을 이용하면 시계가 가리키는 시간이나 장치가 부팅된 이후 경과된 시간에 기반해 서비스를 시작하거나, 액티비티를 열거나, 인텐트를 방송할 수 있다. 알람은 자신을 소유한 애플리케이션이 종료한 뒤에도 발생되며, (필요한 경우) 장치를 절전 상태에서 깨울 수 있다.

## 서비스 소개

풍부한 그래픽 인터페이스를 선사하는 액티비티와는 달리, 서비스는 백그라운드에서 실행되면서 콘텐트 프로바이더를 업데이트하고, 인텐트를 발생시키고, 알림을 만들어내는 일을 한다. 애플리케이션의 액티비티가 화면에서 사라지거나, 비활성화되거나, 종료된 경우에도 지속적인 처리나 규칙적인 처리 혹은 이벤트 처리를 수행해야 한다면 서비스가 답이다.

서비스는 다른 애플리케이션 컴포넌트에서 시작, 중지, 제어된다. 여기서 말한 애플리케이션 컴포넌트에는 다른 서비스, 액티비티, 브로드캐스트 리시버 등이 포함된다. 애플리케이션이 사용자의 입력과 직접적인 의존관계가 없는 기능을 수행하는 경우에도 서비스가 답이다.

시작된 서비스는 비활성 액티비티나 화면에 보이지 않는 액티비티보다 항상 높은 우선순위를 부여받는다. 때문에 실행 중인 서비스가 런타임의 리소스 관리에 의해 종료될 가능성은 낮다. 안드로이드는 서비스를 중지시키는 것만이 포그라운드 컴포넌트(보통은 액티비티)가 필요로 하는 리소스를 제공할 수 있는 유일한 길인 경우에만 서비스를 중지시킨다. 이렇게 중지된 서비스는 리소스가 충분해지면 자동으로 재시작된다.

서비스가 사용자와 직접 상호작용하고 있는 경우(예를 들어, 사용자가 선택한 음악을 재생하는 경우가 있을 수 있다)에는 서비스의 우선순위를 포그라운드 액티비티만큼 높여야 할지 모른다. 이렇게 하면 극단적인 경우를 제외한 나머지 상황에서 서비스가 종료되는 것을 막을 수 있지만, 런타임의 리소스 관리 능력을 축소시켜 전체적인 사용자 경험이 저하될 수 있다.

주기적으로 뭔가를 업데이트하지만 사용자와의 상호작용을 거의 필요로 하지 않는 애플리케이션은 서비스로 구현하기 좋은 후보다. MP3 플레이어와 스포츠 점수 모니터가 바로 화면에 보이는 액티비티 없이 계속 실행되고 업데이트되어야 하는 애플리케이션의 예다.

안드로이드 소프트웨어 스택 안에서도 이러한 예들을 찾아볼 수 있다. 안드로이드는 위치 매니저, 미디어 컨트롤러, 알림 매니저 등을 서비스로 구현해두고 있다.

## 서비스 만들고 제어하기

이번 절에서는 새로운 서비스를 만드는 법과 인텐트와 `startService` 메서드를 이용해 서비스를 시작하고 중지하는 법을 배운다. 그런 다음, 서비스와 액티비티를 바인드해 보다 풍부한 통신 인터페이스를 제공하는 법을 배운다.

### 서비스 만들기

서비스를 정의하려면 `Service`를 확장하는 새로운 클래스를 만든다. 그런 다음, 코드 9-1에서 보이는 것처럼 `onBind`와 `onCreate`를 재정의해야 한다.


**코드 9-1** 서비스 클래스의 골격 코드

```
import android.app.Service;
import android.content.Intent;
```

```
import android.os.IBinder;

public class MyService extends Service {

    @Override
    public void onCreate() {
        // TODO: 서비스 생성 시 수행할 동작.
    }

    @Override
    public IBinder onBind(Intent intent) {
        // TODO: 서비스 바인딩 구현으로 대체한다.
        return null;
    }
}
```

대부분의 경우 onStartCommand도 재정의하고자 할 것이다. 이 메서드는 서비스가 startService 호출을 통해 시작될 때마다 호출되므로, 서비스 수명 안에서 여러 번 실행될 수 있다. 여러분의 서비스가 이 사실을 감안하고 있는지 확실히 해야 한다.

onStartCommand 이벤트 핸들러는 안드로이드 2.0 이전에 사용되던 onStart를 대체한다. onStartCommand는 onStart와는 달리, 서비스가 명시적으로 stopService나 stopSelf를 호출하기 전에 시스템이 서비스를 종료하는 경우에 시스템에게 재시작을 어떻게 처리해야 하는지 알려줄 수 있다.

코드 9-1을 확장하는 다음의 코드는 onStartCommand 핸들러를 재정의하기 위한 골격 코드를 보여준다. 값을 리턴하고 있음을 눈여겨보자. 이 값은 서비스가 런타임에 의해 종료된 뒤 재시작되는 경우에 시스템이 어떻게 반응해야 하는지를 제어한다.

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    // TODO: 처리를 수행할 백그라운드 스레드를 띄운다.
    return Service.START_STICKY;
}
```

서비스는 애플리케이션의 메인 스레드에서 시작된다. 즉, onStartCommand 핸들러에서 수행되는 모든 처리가 메인 GUI 스레드에서 일어나게 된다는 의미다. 서비스를 구현하는 표준 패턴은 onStartCommand에서 새로운 스레드를 만들어 실행해 서비스에서 처리해야 할 작

업들을 백그라운드에서 수행하고, 작업이 완료되면 서비스를 중지하는 것이다(백그라운드 스레드를 만들고 관리하는 법은 이 장 후반부에서 살펴본다).

이 패턴은 onStartCommand가 빨리 리턴할 수 있게 해주며, Service에 정의된 다음의 상수들을 이용해 재시작 방식을 제어할 수 있게 해준다.

- START_STICKY 표준 방식을 말한다. 안드로이드 2.0 이전에 쓰이던 onStart의 구현 방식과 비슷하다. 이 값을 리턴하면 서비스가 런타임에 의해 종료된 뒤 재시작될 때마다 항상 onStartCommand가 호출된다. 단, 재시작 시 onStartCommand에 전달되는 인텐트 매개변수는 null이라는 사실에 유념하자.

  이 모드는 보통 자신만의 상태를 다루는 서비스와 필요에 따라 (startService와 stopService를 통해) 명시적으로 시작되고 중지되는 서비스에 쓰인다. 여기에는 음악을 재생하는 서비스나 지속적인 백그라운드 작업을 처리하는 서비스가 포함된다.

- START_NOT_STICKY 이 모드는 특정 동작이나 명령을 처리하기 위해 시작되는 서비스에 쓰인다. 이런 서비스들은 보통 해당 명령을 처리한 뒤 stopSelf를 이용해 종료한다.

  이 모드로 설정된 서비스가 런타임에 의해 종료되면, 대기 중인 시작 호출pending start calls이 있는 경우에만 재시작된다. 서비스가 종료된 이후 startService를 호출하지 않으면, 해당 서비스는 중지되고 onStartCommand는 호출되지 않는다.

  이 모드는 특정 요청을 처리하는 서비스, 특히 업데이트나 네트워크 폴링network polling 같은 규칙적인 처리를 다루는 서비스에 적합하다. 리소스 경쟁이 벌어지고 있는 동안에는 서비스를 재시작하기보다는 중지된 상태로 놔두고 다음 예약 시점에 다시 시도하는 것이 보다 현명한 방법이다.

- START_REDELIVER_INTENT 어떤 경우에는 서비스가 요청받은 명령을 반드시 처리 완료하도록 하고 싶을 때가 있다.

  이 모드는 앞서 살펴본 두 모드의 조합이다. 서비스가 런타임에 의해 종료되면, 해당 서비스는 대기 중인 시작 호출이 있는 경우나 프로세스가 stopSelf를 호출하기 전에 종료된 경우에만 재시작된다.

  후자의 경우에는 onStartCommand가 호출되며, 제대로 처리되지 않은 초기 인텐트가 전달된다.

각 모드는 서비스가 처리를 완료할 경우 stopService나 stopSelf를 통해 서비스를 명시적으로 중지할 것을 요구한다. 이 두 메서드는 이번 장 후반부에서 보다 자세히 살펴본다.

안드로이드 SDK 2.0 (SDK API 레벨 5) 이전에는 서비스가 시작될 때 필요한 작업을 수행할 수 있도록 서비스 클래스가 onStart 이벤트 핸들러를 호출했었다. 이제 onStart 핸들러를 구현하는 것은 onStartCommand를 재정의해 START_STICKY 플래그를 리턴하도록 구현하는 것과 같다.

onStartCommand의 리턴 값으로 지정하는 재시작 모드는 이후 이어지는 호출에 전달되는 매개변수 값에 영향을 미친다.

처음에는 서비스를 시작하기 위해 startService에 전달한 인텐트가 매개변수로 전달된다. 그러다 서비스가 재시작되면 START_STICKY 모드의 경우 null이 전달되며, START_REDELIVER_INTENT 모드의 경우 원본 인텐트가 전달된다.

flag 매개변수를 이용하면 서비스가 어떻게 시작됐는지를 알아낼 수 있다. 특히 코드 9-2에 있는 코드를 이용하면 다음 두 가지 경우 중 어느 것이 참인지를 판별할 수 있다.

- START_FLAG_REDELIVERY 매개변수로 전달된 인텐트가 재전송된 인텐트임을 나타낸다. 이 재전송은 서비스가 명시적으로 stopSelf를 호출하기 전에 시스템 런타임이 서비스를 종료한 경우 발생한다.
- START_FLAG_RETRY 서비스가 비정상적인 종료 후 재시작되었음을 나타낸다. 서비스가 START_STICKY로 설정된 경우 전달된다.


Available for download on Wrox.com

**코드 9-2** 서비스의 시작 이유 판별하기

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    if ((flags & START_FLAG_RETRY) == 0) {
        // TODO: 재시작 된경우에 원하는 작업을 수행한다.
    }
    else {
        // TODO: 처리를 수행할 백그라운드 스레드를 띄운다.
    }
    return Service.START_STICKY;
}
```

### 서비스를 매니페스트에 등록하기

새로운 서비스를 만들고 난 뒤에는 이를 애플리케이션 매니페스트에 등록해야 한다.

서비스 등록은 application 노드에 <service> 태그를 포함시킴으로써 이뤄진다. requires-permission 속성을 이용하면 다른 애플리케이션에서 이 서비스를 접근하기 위해 필요한 권한을 지정할 수 있다.

다음은 앞에서 만든 서비스를 등록하기 위한 service 태그다.

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

### 서비스 스스로 종료하기

서비스는 주어진 동작이나 처리를 완료한 뒤 stopSelf를 호출해야 한다. stopSelf를 매개변수 없이 호출하면 서비스가 강제 중지된다. 지금까지 호출된 각 startService에 대한 처리가 완료되도록 보장하려면, 아래에 보이는 것처럼 startId 값을 가지고 stopSelf를 호출한다.

```
stopSelf(startId);
```

이처럼 처리를 마친 뒤 서비스를 명시적으로 중지하면 시스템이 실행에 필요한 리소스를 회수할 수 있게 된다. 서비스는 우선순위가 높기 때문에 런타임에 의해 종료되는 경우가 드물다. 따라서 서비스가 스스로 종료하면 여러분의 애플리케이션이 사용하는 리소스의 양을 크게 줄일 수 있다.

### 서비스 시작하고, 제어하고, 상호작용하기

서비스를 시작하려면 startService를 호출한다. 액션을 이용해 적절한 브로드캐스트 리시버를 가진 서비스를 암시적인 방법으로 시작할 수도 있고, 원하는 서비스의 클래스를 이용해 해당 서비스를 명시적으로 지정할 수도 있다. 서비스가 애플리케이션이 가지고 있지 않은 권한을 요구하면, startService 호출은 SecurityException을 던질 것이다.

두 경우 모두 인텐트에 엑스트라를 추가하는 방법으로 서비스의 onStart 핸들러에 값을 전달할 수 있다. 코드 9-3은 서비스를 시작하는 데 쓰이는 두 가지 기법을 설명한다.


Available for download on Wrox.com

**코드 9-3** 서비스 시작하기

```
// 암시적인 방법으로 서비스를 시작한다.
Intent myIntent = new Intent(MyService.ORDER_PIZZA);
myIntent.putExtra("TOPPING", "Margherita");
startService(myIntent);

// 명시적인 방법으로 서비스를 시작한다.
startService(new Intent(this, MyService.class));
```

이 예제를 이용하려면 MyService 클래스 안에 ORDER_PIZZA라는 상수를 포함시키고, 인텐트 필터를 이용해 이 서비스를 ORDER_PIZZA의 프로바이더로 등록해야 할 것이다.

서비스를 중지하려면 중지하려는 서비스를 정의하고 있는 인텐트를 가지고 stopService를 호출한다. 코드 9-4는 먼저 서비스를 시작한 다음, startService 호출 시 리턴되는 컴포넌트 이름을 이용하는 방법과 명시적인 방법으로 서비스를 중지한다.


Available for download on Wrox.com

**코드 9-4** 서비스 중지하기

```
ComponentName service = startService(new Intent(this, BaseballWatch.class));
// 서비스 이름으로 서비스 중지한다.
stopService(new Intent(this, service.getClass()));
// 명시적으로 서비스를 중지한다.
try {
    Class serviceClass = Class.forName(service.getClassName());
    stopService(new Intent(this, serviceClass));
} catch (ClassNotFoundException e) {}
```

이미 실행 중인 서비스에 대해 startService를 호출하면 해당 서비스의 onStartCommand 핸들러가 다시 실행된다. startService는 중첩 호출되지 않는다. 때문에 startService가 몇 번이나 호출됐는지에 관계없이 한 번만 stopService를 호출하면 서비스가 종료한다.

### 지진 정보 모니터링 서비스 예제

이번 장에서는 5장에서부터 만들기 시작하여 6장, 7장, 8장을 통해 계속해서 기능을 추가해 온 지진 정보 예제를 수정한다. 이번 예제에서는 지진 정보를 업데이트하고 처리하는 기능을 별도의 서비스 컴포넌트로 옮긴다.

이 장 후반부에서는 네트워크 조회와 XML 파싱을 백그라운드 스레드로 옮기는 것을 시작으로, 토스트와 알림을 이용해 사용자에게 새로운 지진 정보를 알리는 등 이 서비스 안에 추가 기능들을 만들 것이다.

**1.** 먼저 Service를 확장하는 EarthquakeService라는 이름의 새로운 클래스를 만든다.

```
package com.paad.earthquake;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import java.util.Timer;
import java.util.TimerTask;

public class EarthquakeService extends Service {
    @Override
    public void onCreate() {
        // TODO: 변수들을 초기화하고, GUI 객체들의 레퍼런스를 얻어온다.
    }

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

**2.** 애플리케이션 매니페스트의 application 노드에 service 태그를 이용하여 이 서비스를 추가한다.

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

**3.** Earthquake 액티비티의 refreshEarthquakes와 addNewQuake 메서드를 EarthquakeService로 옮긴다.

이어서 addQuakeToArray와 loadQuakesFromProvider에 대한 호출을 제거한다(이들 메서드 자체는 여전히 필요하므로 Earthquake 액티비티에 남겨둔다). EarthquakeService에서 earthquakes 배열 리스트를 참조하는 코드도 모두 제거한다.

```
private void addNewQuake(Quake _quake) {
    ContentResolver cr = getContentResolver();

    // 이 지진 정보가 콘텐트 프로바이더에 이미 존재하고 있지는 않은지
    // 확인하기 위한 where 절을 구성한다.
    String w = EarthquakeProvider.KEY_DATE + " = " +
               _quake.getDate().getTime();

    // 새로운 지진 정보라면 콘텐트 프로바이더에 넣는다.
    Cursor c = cr.query(EarthquakeProvider.CONTENT_URI,
                        null, w, null, null);
    if (c.getCount()==0){
        ContentValues values = new ContentValues();

        values.put(EarthquakeProvider.KEY_DATE, _quake.getDate().getTime());
        values.put(EarthquakeProvider.KEY_DETAILS, _quake.getDetails());
        double lat = _quake.getLocation().getLatitude();
        double lng = _quake.getLocation().getLongitude();
        values.put(EarthquakeProvider.KEY_LOCATION_LAT, lat);
        values.put(EarthquakeProvider.KEY_LOCATION_LNG, lng);
        values.put(EarthquakeProvider.KEY_LINK, _quake.getLink());
        values.put(EarthquakeProvider.KEY_MAGNITUDE, _quake.getMagnitude());
        cr.insert(EarthquakeProvider.CONTENT_URI, values);
    }
    c.close();
}

private void refreshEarthquakes() {
    // XML을 가져온다.
    URL url;
    try {
        String quakeFeed = getString(R.string.quake_feed);
        url = new URL(quakeFeed);

        URLConnection connection;
        connection = url.openConnection();

        HttpURLConnection httpConnection =
            (HttpURLConnection)connection;
        int responseCode = httpConnection.getResponseCode();

        if (responseCode == HttpURLConnection.HTTP_OK) {
            InputStream in = httpConnection.getInputStream();
```

```
DocumentBuilderFactory dbf =
DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();

// 지진 정보 피드를 파싱한다.
Document dom = db.parse(in);
Element docEle = dom.getDocumentElement();

// 지진 정보로 구성된 리스트를 얻어온다.
NodeList nl = docEle.getElementsByTagName("entry");
if (nl != null && nl.getLength() > 0) {
    for (int i = 0 ; i < nl.getLength(); i++) {
        Element entry = (Element)nl.item(i);
        Element title;
        title =
            (Element)entry.getElementsByTagName("title").item(0);
        Element g =
            (Element)entry.getElementsByTagName("georss:point").item(0);
        Element when =
            (Element)entry.getElementsByTagName("updated").item(0);
        Element link =
            (Element)entry.getElementsByTagName("link").item(0);

        String details = title.getFirstChild().getNodeValue();
        String hostname = "http://earthquake.usgs.gov";
        String linkString = hostname + link.getAttribute("href");

        String point = g.getFirstChild().getNodeValue();
        String dt = when.getFirstChild().getNodeValue();
        SimpleDateFormat sdf;
        sdf = new SimpleDateFormat("yyyy-MM-dd'T'hh:mm:ss'Z'");
        Date qdate = new GregorianCalendar(0,0,0).getTime();
        try {
            qdate = sdf.parse(dt);
        } catch (ParseException e) {
            e.printStackTrace();
        }

        String[] location = point.split(" ");
        Location l = new Location("parsed");
        l.setLatitude(Double.parseDouble(location[0]));
        l.setLongitude(Double.parseDouble(location[1]));

        String magnitudeString = details.split(" ")[1];
```

```
                    int end = magnitudeString.length()-1;
                    double magnitude =
                        Double.parseDouble(magnitudeString.substring(0, end));

                    details = details.split(",")[1].trim();

                    Quake quake = new Quake(qdate, details, l, magnitude,
                                            linkString);

                    // 새로운 지진 정보를 처리한다.
                    addNewQuake(quake);
                }
            }
        }
    } catch (MalformedURLException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (ParserConfigurationException e) {
        e.printStackTrace();
    } catch (SAXException e) {
        e.printStackTrace();
    }
    finally {
    }
}
```

**4.** Earthquake 액티비티에 refreshEarthquakes 메서드를 새로 만든다. 이 메서드는 EarthquakeService를 명시적으로 시작시키는 일을 한다.

```
private void refreshEarthquakes() {
    startService(new Intent(this, EarthquakeService.class));
}
```

**5.** 다시 EarthquakeService로 돌아온다. 이번에는 onStartCommand와 onCreate 메서드를 재정의해 지진 정보 리스트를 업데이트하는 데 사용할 새로운 타이머를 지원하도록 구현한다. 우리는 타이머를 이용해 지진 정보를 업데이트하므로 onStartCommand는 START_STICKY를 리턴해야 한다. 사실 이러한 방식은 일반적으로 바람직하지 않다. 이렇게 하기보다는 타이머로 처리하고 있는 동작들을 백그라운드 스레드로 옮기고, 알람을 통해 실행되도록 구현해야 한다. 이 방식에 대해서는 이번 장 후반부에서 배울 것이다.

지진 정보가 주기적으로 업데이트되어야 하는지의 여부는 6장에서 만든 SharedPreferences 객체로 판별한다.❶

```
private Timer updateTimer;

private int minimumMagnitude = 0;
private boolean autoUpdate = false;
private int updateFreq = 0;

@Override
public int onStartCommand(Intent intent, int flags, int startId) {

    // 공유 환경설정을 얻어온다.
    Context context = getApplicationContext();
    SharedPreferences prefs =
        PreferenceManager.getDefaultSharedPreferences(context);

    autoUpdate =
        prefs.getBoolean(Preferences.PREF_AUTO_UPDATE, false);

    minimumMagnitude =
        Integer.parseInt(prefs.getString(Preferences.PREF_MIN_MAG, "0"));

    updateFreq =
        Integer.parseInt(prefs.getString(Preferences.PREF_UPDATE_FREQ, "0"));

    updateTimer.cancel();

    if(autoUpdate) {
        updateTimer = new Timer("earthquakeUpdates");

        updateTimer.scheduleAtFixedRate(new TimerTask() {
            public void run() {
                refreshEarthquakes();
            }
        }, 0, updateFreq*60*1000);
    }
```

❶ **옮긴이** 아래의 import 문도 추가한다.

```
import android.content.SharedPreferences;
import android.content.res.Resources;
import android.preference.PreferenceManager;
```

```
        refreshEarthquakes();

        return Service.START_STICKY;
    };

    @Override
    public void onCreate() {
        updateTimer = new Timer("earthquakeUpdates");
    }
```

6. 이제 EarthquakeService는 업데이트 요청이 있을 때뿐만 아니라 자동 업데이트가 체크되어 있는 경우에도 설정된 업데이트 빈도에 따라 지진 정보 프로바이더를 업데이트할 것이다. 그러나 아직 업데이트된 정보를 Earthquake 액티비티의 리스트 뷰나 EarthquakeMap 액티비티에 전달하지는 않았다.

   EarthquakeService를 수정해 새로운 지진 정보가 추가될 때마다 인텐트를 방송하도록 구현하자. 이렇게 하면 앞서 언급한 컴포넌트들뿐만 아니라 지진 정보 데이터에 관심 있는 기타 다른 애플리케이션들에게도 정보가 업데이트되었음을 알려줄 수 있다.

   6.1. addNewQuake 메서드를 수정해 announceNewQuake라는 이름의 새로운 메서드를 호출하도록 구현한다.

```
public static final String NEW_EARTHQUAKE_FOUND = "New_Earthquake_Found";

private void addNewQuake(Quake _quake) {
    ContentResolver cr = getContentResolver();

    // 이 지진 정보가 콘텐트 프로바이더에 이미 존재하고 있지는 않은지
    // 확인하기 위한 where 절을 구성한다.
    String w = EarthquakeProvider.KEY_DATE +
               " = " + _quake.getDate().getTime();

    // 새로운 지진 정보라면 콘텐트 프로바이더에 넣는다.
    if (cr.query(EarthquakeProvider.CONTENT_URI,
```

```
                null, w, null, null).getCount()==0){
            ContentValues values = new ContentValues();

            values.put(EarthquakeProvider.KEY_DATE, _quake.getDate().getTime());
            values.put(EarthquakeProvider.KEY_DETAILS, _quake.getDetails());

            double lat = _quake.getLocation().getLatitude();
            double lng = _quake.getLocation().getLongitude();
            values.put(EarthquakeProvider.KEY_LOCATION_LAT, lat);
            values.put(EarthquakeProvider.KEY_LOCATION_LNG, lng);
            values.put(EarthquakeProvider.KEY_LINK, _quake.getLink());
            values.put(EarthquakeProvider.KEY_MAGNITUDE,
                       _quake.getMagnitude());

            cr.insert(EarthquakeProvider.CONTENT_URI, values);
            announceNewQuake(_quake);
        }
    }

    private void announceNewQuake(Quake quake) {
    }
```

**6.2.** announceNewQuake를 수정해 새로운 지진 정보가 추가될 때마다 인텐트를 방송하도록 구현한다.

```
private void announceNewQuake(Quake quake) {
    Intent intent = new Intent(NEW_EARTHQUAKE_FOUND);
    intent.putExtra("date", quake.getDate().getTime());
    intent.putExtra("details", quake.getDetails());
    intent.putExtra("longitude", quake.getLocation().getLongitude());
    intent.putExtra("latitude", quake.getLocation().getLatitude());
    intent.putExtra("magnitude", quake.getMagnitude());

    sendBroadcast(intent);
}
```

**7.** 이것으로 EarthquakeService 구현을 마무리한다. 이제 Earthquake 프로젝트에 있는 액티비티들을 수정해 announceNewQuake에서 방송하는 인텐트를 받아 적절히 화면을 업데이트하도록 구현하는 일이 남았다.

**7.1.** Earthquake 액티비티 안에 BroadcastReceiver를 확장하는 EarthquakeReceiver라는 이름의 새로운 내부 클래스를 만든다. 이어서 onReceive 메서드를

재정의하고 loadQuakesFromProvider를 호출하도록 구현해 earthquakes 배열을 업데이트하고 리스트 뷰를 갱신한다.❷

```
public class EarthquakeReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        loadQuakesFromProvider();
    }
}
```

**7.2.** onResume 메서드를 재정의해 액티비티가 활성화되면 단계 7.1에서 만든 새로운 브로드캐스트 리시버를 등록하고 리스트 뷰의 내용을 업데이트하도록 구현한다. 또한 onPause 메서드를 재정의해 액티비티가 포그라운드 밖으로 이동하면, onResume에서 등록한 브로드캐스트 리시버를 등록 해제하도록 구현한다.❸

```
EarthquakeReceiver receiver;

@Override
public void onResume() {
    IntentFilter filter;
    filter = new IntentFilter(EarthquakeService.NEW_EARTHQUAKE_FOUND);
    receiver = new EarthquakeReceiver();
    registerReceiver(receiver, filter);

    loadQuakesFromProvider();
    super.onResume();
}

@Override
public void onPause() {
    unregisterReceiver(receiver);
    super.onPause();
}
```

❷ **옮긴이** 아래의 import 문도 추가한다.

```
import android.content.BroadcastReceiver;
```

❸ **옮긴이** 아래의 import 문도 추가한다.

```
import android.content.IntentFilter;
```

**7.3.** EarthquakeMap 액티비티에도 동일한 작업을 수행하는데, 이번에는 인텐트가 수신될 때마다 결과 커서에 requery를 호출하고 MapView를 무효화한다.❹

```
EarthquakeReceiver receiver;

@Override
public void onResume() {
    earthquakeCursor.requery();

    IntentFilter filter;
    filter = new IntentFilter(EarthquakeService.NEW_EARTHQUAKE_FOUND);
    receiver = new EarthquakeReceiver();
    registerReceiver(receiver, filter);

    super.onResume();
}

@Override
public void onPause() {
    earthquakeCursor.deactivate();
    super.onPause();
}

public class EarthquakeReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        earthquakeCursor.requery();
        MapView earthquakeMap = (MapView)findViewById(R.id.map_view);
        earthquakeMap.invalidate();
    }
}
```

이제 Earthquake 액티비티는 자신이 실행될 때마다 지진 정보 서비스를 시작한다. 이 서비스는 Earthquake 액티비티가 일시 중단되거나 종료된 뒤에도 백그라운드에서 계속 실행되면서 지진 정보 콘텐트 프로바이더를 업데이트할 것이다.

❹ **옮긴이** 아래의 import 문도 추가한다.

```
import android.content.BroadcastReceiver;
import android.content.IntentFilter;
```

앞으로 남은 부분에서는 토스트, 알림, 알람을 이용해 이 지진 정보 서비스를 계속해서 업그레이드하고 발전시켜나갈 것이다.

현재 지진 정보 처리 작업은 서비스에서 수행되지만, 서비스는 여전히 메인 GUI 스레드에서 실행되고 있다. 시간이 많이 드는 작업을 백그라운드 스레드로 옮기면 성능이 향상되고 "강제 종료Force Close" 메시지도 피할 수 있다. 이에 대해서는 이번 장 후반부에서 살펴본다.

또한 이 지진 정보 서비스는 소중한 리소스를 차지하면서 끊임없이 실행된다. 다음 절에서는 이 지진 정보 서비스에서 사용하고 있는 타이머를 알람으로 대체하는 법을 설명한다.

## 서비스에 액티비티 바인드하기

액티비티를 서비스에 바인드하면, 액티비티는 해당 서비스의 레퍼런스를 얻어올 수 있게 된다. 이 레퍼런스를 이용하면 실행 중인 서비스에 대해 메서드를 호출할 수 있다. 서비스를 인스턴스화된 클래스처럼 다룰 수 있게 되는 것이다.

액티비티가 서비스와의 보다 긴밀한 소통을 필요로 한다면 바인딩을 이용하자. 서비스가 바인딩을 지원하기 위해서는 코드 9-5에서 보이는 것처럼 `onBind` 메서드를 구현하고 있어야 한다.

**코드 9-5** 서비스에 바인딩 기능 구현하기

Available for download on Wrox.com

```
private final IBinder binder = new MyBinder();

@Override
public IBinder onBind(Intent intent) {
    return binder;
}

public class MyBinder extends Binder {
    MyService getService() {
        return MyService.this;
    }
}
```

서비스와 액티비티 간의 연결은 `ServiceConnection`으로 표현된다. 연결이 수립된 서비스 인스턴스의 레퍼런스를 얻어오려면, 코드 9-6에서 보이는 것처럼 `ServiceConnection`을

구현하고 onServiceConnected와 onServiceDisconnected 메서드를 재정의해야 한다.


**코드 9-6** 서비스에 바인드하기

```
// 서비스에 대한 레퍼런스.
private MyService serviceBinder;

// 서비스와 액티비티 간의 연결을 다룬다.
private ServiceConnection mConnection = new ServiceConnection() {
    public void onServiceConnected(ComponentName className, IBinder service) {
        // 연결이 이뤄지면 호출된다.
        serviceBinder = ((MyService.MyBinder)service).getService();
    }

    public void onServiceDisconnected(ComponentName className) {
        // 연결이 예기치 않게 끊어진 경우 호출된다.
        serviceBinder = null;
    }
};
```

바인딩을 수행하려면 bindService를 호출한다. 매개변수에는 바인드할 서비스가 명시적 또는 암시적으로 지정된 인텐트와 여러분이 구현한 ServiceConnection의 인스턴스를 전달한다. 코드 9-6에 이어 아래의 코드를 참고하자.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // 서비스에 바인드한다.
    Intent bindIntent = new Intent(MyActivity.this, MyService.class);
    bindService(bindIntent, mConnection, Context.BIND_AUTO_CREATE);
}
```

서비스가 바인드되고 나면, onServiceConnected 핸들러를 통해 얻은 서비스 객체(serviceBinder)를 통해 해당 서비스가 가진 모든 public 메서드와 속성을 사용할 수 있게 된다.

안드로이드 애플리케이션은 보통 메모리를 공유하지 않지만, 경우에 따라서는 다른 애플리케이션 프로세스에서 실행 중인 서비스와 바인드해 상호작용하고 싶을 수도 있다.

브로드캐스트 인텐트나 서비스를 시작시키는 데 사용되는 인텐트의 엑스트라 번들을 이용하면, 다른 프로세스에서 실행 중인 서비스와 통신할 수 있다. 보다 단단히 결합된 연결을 원한다면 AIDL을 이용할 수 있다. AIDL을 이용하면 애플리케이션 경계를 넘어 바인딩될 수 있는 서비스를 만들 수 있다. AIDL은 서비스의 인터페이스를 OS 수준에서 정의하기 때문에 안드로이드가 프로세스 경계 너머로 객체를 전송할 수 있게 해준다. AIDL 정의는 15장에서 다룬다.

## 백그라운드 서비스의 우선순위 높이기

3장에서도 배웠다시피 안드로이드는 동적인 방식으로 리소스를 관리한다. 때문에 안드로이드에서는 애플리케이션, 액티비티, 서비스가 런타임에 의해 아무런 경고 없이 종료될 수 있다.

종료시킬 애플리케이션과 애플리케이션 컴포넌트를 결정할 때, 안드로이드는 실행 중인 서비스에게 두 번째로 높은 우선순위를 부여한다. 오직 활성화된 상태의 포그라운드 액티비티만이 시스템 리소스 측면에서 가장 우선순위가 높은 것으로 간주된다.

서비스가 사용자와 직접 상호작용하고 있는 극단적인 경우에는 서비스의 우선순위가 포그라운드 액티비티만큼 높아질 수도 있다. `startForeground` 메서드를 이용해 서비스가 포그라운드에서 실행되도록 설정하는 경우가 바로 이 경우에 해당한다.

포그라운드에서 실행 중인 서비스는 아마도 사용자와 직접 상호작용하고 있을 것이다(이를테면, 사용자가 선택한 음악을 재생하고 있는 경우가 있을 수 있다). 때문에 사용자는 항상 포그라운드 서비스의 존재를 인지할 수 있어야 한다. 이를 보장하기 위해 `startForeground`를 호출할 때는 코드 9-7에서 보이는 것처럼 반드시 진행 중 알림ongoing Notification(이에 대해서는 이번 장 후반부에서 보다 자세히 설명한다)을 지정해야 한다. 이 알림은 해당 서비스가 포그라운드에서 실행되고 있는 한 계속 이어질 것이다.

서비스를 포그라운드로 옮기면, 런타임은 사실상 리소스 회수를 목적으로 해당 서비스를 종료할 수 없게 된다. 이처럼 종료할 수 없는 서비스들이 동시에 여러 개 실행되고 있으면, 시스템이 리소스 부족 상황에서 복구되기가 극도로 어려워질 수 있다.

이 기법은 여러분의 서비스가 제 기능을 하기 위해 꼭 필요한 경우에만 사용하자. 또 서비스를 포그라운드로 옮길 때는 꼭 필요한 시간만큼만 머물러 있게 하자.


**코드 9-7** 서비스를 포그라운드로 옮기기

```
int NOTIFICATION_ID = 1;

Intent intent = new Intent(this, MyActivity.class);
PendingIntent pi = PendingIntent.getActivity(this, 1, intent, 0));

Notification notification = new Notification(R.drawable.icon,
    "포그라운드에서 실행 중 ", System.currentTimeMillis());
notification.setLatestEventInfo(this, "제목", "텍스트", pi);

notification.flags = notification.flags ¦
                     Notification.FLAG_ONGOING_EVENT;

startForeground(NOTIFICATION_ID, notification);
```

코드 9-7은 setLatestEventInfo를 이용해 기본 상태 윈도우 레이아웃으로 알림을 업데이트한다. 알림에 커스텀 레이아웃을 지정하는 법은 이번 장 후반부에서 배운다. 알림에 커스텀 레이아웃을 적용하면, 사용자에게 현재 실행 중인 서비스에 대한 보다 자세한 정보를 제공할 수 있다.

서비스가 더 이상 포그라운드 우선순위를 필요로 하지 않으면 이를 다시 백그라운드로 옮길 수 있으며, 옵션으로 코드 9-8에서처럼 stopForeground 메서드를 이용해 해당 서비스의 진행 중 알림을 제거할 수 있다. 서비스가 중지하거나 종료하면 알림은 자동으로 취소된다.


**코드 9-8** 서비스를 다시 백그라운드로 옮기기

```
// 백그라운드로 이동하고 알림을 제거한다.
stopForeground(true);
```

안드로이드 2.0 이전에는 setForeground 메서드를 이용해 서비스를 포그라운드로 설정하는 것이 가능했다. 현재 이 메서드는 폐기 대상이 되었으며 아무 일도 하지 않는다.

## 백그라운드 스레드 이용하기

애플리케이션이 좋은 반응성을 유지하기 위해서는 느리고 시간이 많이 드는 모든 작업을 메인 애플리케이션 스레드에서 자식 스레드로 옮기는 것이 좋다.

액티비티, 서비스, 브로드캐스트 리시버 등 안드로이드의 모든 애플리케이션 컴포넌트들은 메인 애플리케이션 스레드에서 시작된다. 따라서 이들 컴포넌트가 시간이 많이 드는 처리를 수행한다면, 서비스와 화면에 보이는 액티비티를 포함한 다른 모든 컴포넌트들이 블록될 것이다.

안드로이드는 이러한 처리 작업을 백그라운드로 옮기기 위한 두 가지 대안을 제공한다. AsyncTask 클래스는 백그라운드에서 수행할 작업을 정의할 수 있게 해주며, 작업 진행 상황을 모니터하고 결과를 GUI 스레드로 보내는 데 사용할 수 있는 이벤트 핸들러들을 제공한다.

아니면 여러분만의 Thread를 구현하고 Handler 클래스로 GUI 스레드와 동기화한 뒤 UI를 업데이트할 수도 있다. 두 기법 모두 이번 절에서 설명한다.

2장에서 설명한 "강제 종료" 다이얼로그 박스를 피하기 위해서는 백그라운드 스레드 사용이 필수다. 안드로이드는 (키 누름 같은) 입력 이벤트에 대해 5초 이내로 반응하지 않는 액티비티와 onReceive 핸들러를 10초 이내에 완료하지 못하는 브로드캐스트 리시버를 반응이 느리다고 간주한다.

이러한 상황에 처해 애플리케이션이 종료되길 원하는 사람은 아무도 없을 것이다. 파일 작업, 네트워크 조회, 데이터베이스 트랜잭션, 복잡한 계산 등 시간이 많이 드는 모든 처리에는 백그라운드 스레드를 이용하자.

### AsyncTask로 비동기 작업 수행하기

AsyncTask 클래스는 시간이 많이 드는 작업을 백그라운드 스레드로 옮기기 위한 간단하고 편리한 메커니즘을 제공한다. AsyncTask 클래스는 GUI 스레드와 동기화되어 있는 편리한 이벤트 핸들러들을 제공한다. 이를 이용하면 작업 진행 상황을 보고하거나 작업 완료 시 결과를 표시하기 위해 뷰와 다른 UI 요소들을 업데이트할 수 있다.

AsyncTask는 스레드의 생성, 관리, 동기화와 관련된 모든 일을 처리한다. AsyncTask를 이용하면 백그라운드에서 수행할 비동기 작업을 생성할 수 있고, 작업 완료 시 UI를 업데이트할 수 있다.

### 새로운 비동기 작업 만들기

새로운 비동기 작업을 만들기 위해서는 코드 9-9에서 보이는 것처럼 AsyncTask를 확장해야 한다. 이때 AsyncTask에는 아래의 형식에 따라 execute 메서드의 입력 매개변수, 진행 상황 보고 값, 결과 값으로 쓰일 클래스들을 지정해야 한다.

```
AsyncTask<[입력 매개변수 타입], [진행 상황 보고 타입], [결과 타입]>
```

입력 매개변수 전달, 진행 상황 업데이트, 최종 결과 보고 중 필요치 않은 것이 있으면 해당 타입에 간단히 Void를 지정하면 된다.


**코드 9-9** 입력 매개변수 타입에는 String을, 진행 상황 보고 및 결과 타입에는 Integer를 사용하는 AsyncTask의 골격 구현

```
private class MyAsyncTask extends AsyncTask<String, Integer, Integer> {
    @Override
    protected void onProgressUpdate(Integer... progress) {
        // [ ... 프로그래스 바나 알림 또는 다른 UI 요소들을 업데이트한다 ... ]
    }

    @Override
    protected void onPostExecute(Integer... result) {
        // [ ... UI를 업데이트하거나 다이얼로그 또는 알림을 통해 결과를 보고한다 ... ]
    }

    @Override
    protected Integer doInBackground(String... parameter) {
        int myProgress = 0;

        // [ ... 백그라운드 처리 작업을 수행하고, myProgress를 업데이트한다 ... ]
        publishProgress(myProgress);
        // [ ... 백그라운드 처리 작업을 계속 수행한다 ... ]

        // onPostExecute에 전달할 값을 리턴한다.
        return result;
    }
}
```

코드 9-9에서 보이는 것처럼 AsyncTask를 확장한 클래스는 다음 이벤트 핸들러들을 구현해야 한다.

- doInBackground AsyncTask의 "입력 매개변수 타입" 부분에 지정되어 있는 타입으로 된 일련의 매개변수들을 전달받는다. 이 메서드는 백그라운드 스레드에서 실행된다. 따라서 이 메서드에서는 절대로 UI 객체와 상호작용하려 해서는 안 된다.

  오래 걸리는 코드를 이 메서드 안에 두고 publishProgress 메서드를 이용하여 onProgressUpdate가 진행 상황을 UI에 전달할 수 있게 한다.

  백그라운드 작업이 완료되면 최종 결과를 onPostExecute 핸들러에 리턴하여 이를 UI에 보고한다.

- onProgressUpdate 중간 진행 상황을 UI 스레드에 전달하려면 이 핸들러를 재정의한다. 이 핸들러는 doInBackground에서 publishProgress에 전달한 일련의 매개변수들을 전달받는다.

  이 핸들러는 실행 시 GUI 스레드와 동기화된다. 따라서 이 메서드에서는 UI 요소들을 안전하게 변경할 수 있다.

- onPostExecute doInBackground가 작업을 마치고 리턴한 값은 이 이벤트 핸들러에 전달된다.

  이 핸들러는 비동기 작업을 마친 후 UI를 업데이트하기 위해 사용한다. 이 핸들러는 실행 시 GUI 스레드와 동기화된다. 따라서 이 메서드에서는 UI 요소들을 안전하게 변경할 수 있다.

### 비동기 작업 실행하기

여러분이 구현한 비동기 작업을 실행하려면, 코드 9-10에서 보이는 것처럼 해당 작업의 새로운 인스턴스를 생성하고 execute를 호출한다. 이때 execute에는 AsyncTask의 "입력 매개변수 타입" 부분에 지정되어 있는 타입으로 된 일련의 매개변수들을 전달할 수 있다.


**코드 9-10** 비동기 작업 실행하기

```
new MyAsyncTask().execute("inputString1", "inputString2");
```

각각의 AsyncTask 인스턴스는 한 번만 실행될 수 있다. 이미 execute가 호출된 AsyncTask 인스턴스에 또 한 번 execute를 호출하면 예외가 발생한다.

## AsyncTask를 이용해 지진 정보 서비스를 백그라운드 스레드로 옮기기

이어지는 예제는 EarthquakeService에서 수행되는 네트워크 조회와 XML 처리를 AsyncTask를 이용해 백그라운드 스레드로 옮기는 법을 보여준다.

**1.** EarthquakeLookupTask라는 이름으로 새로운 AsyncTask를 구현한다. 입력 매개변수 타입과 결과 타입에는 Void를, 진행 상황 보고 타입에는 Quake를 지정한다. 그리고 doInBackground, onProgressUpdate, onPostExecute를 재정의해둔다.[5]

```
public class EarthquakeLookupTask extends AsyncTask<Void, Quake, Void> {
    @Override
    protected Void doInBackground(Void... params) {
        return null;
    }

    @Override
    protected void onProgressUpdate(Quake... values) {
        super.onProgressUpdate(values);
    }

    @Override
    protected void onPostExecute(Void result) {
        super.onPostExecute(result);
    }
}
```

**2.** refreshEarthquakes 메서드에 있는 모든 코드를 doInBackground 핸들러로 옮긴다. 이어서 새로운 지진 정보가 처리될 때마다 publishProgress를 호출하는 코드를 추가한다. publishProgress의 매개변수에는 가장 최근에 파싱된 Quake를 전달한다. 파싱을 완료하면 null을 리턴한다.

```
@Override
protected Void doInBackground(Void... params) {
    [ ... 기존 XML 파싱 코드 ... ]

    // 새로운 지진 정보를 처리한다.
    addNewQuake(quake);
    publishProgress(quake);
```

[5] **옮긴이** 아래의 import 문도 추가한다.

```
import android.os.AsyncTask;
```

```
        [ ... 기존 예외 처리 코드 ... ]

        return null;
    }
```

3. 이제 텅 빈 refreshEarthquakes 메서드를 업데이트한다. 이 메서드는 EarthquakeLookupTask를 생성하고 실행하는 일을 해야 한다. 먼저 이미 시작된 또 다른 비동기 작업이 있는지 확인한다. 업데이트 요청이 쌓이는 것을 피하기 위해 현재 처리되고 있는 작업이 없을 경우에만 업데이트를 시작해야 한다.

```
EarthquakeLookupTask lastLookup = null;

private void refreshEarthquakes() {
    if (lastLookup == null ¦¦
        lastLookup.getStatus().equals(AsyncTask.Status.FINISHED)) {
        lastLookup = new EarthquakeLookupTask();
        lastLookup.execute((Void[])null);
    }
}
```

## 직접 스레드를 만들고 GUI 스레드와 동기화하기

AsyncTask를 이용하면 작업을 쉽고 빠르게 백그라운드로 옮길 수 있다. 하지만 백그라운드 처리를 위한 스레드를 직접 만들어 관리하고 싶을 때도 있다.

이번 절에서는 새로운 스레드 객체를 만들고 시작하는 법과 UI를 업데이트하기에 앞서 GUI 스레드와 동기화하는 법을 배운다.

### 새로운 스레드 만들기

안드로이드의 Handler 클래스와 java.lang.Thread 내에서 이용할 수 있는 스레딩 클래스들을 이용하면 자식 스레드를 만들고 관리할 수 있다. 코드 9-11은 처리를 자식 스레드로 옮기기 위한 간단한 골격 코드를 보여준다.


**코드 9-11** 처리를 백그라운드 스레드로 옮기기

```
// 이 메서드는 메인 GUI 스레드에서 호출된다.
private void mainProcessing() {
    // 이 코드는 시간이 많이 드는 작업을 자식 스레드로 옮긴다.
```

```
    Thread thread = new Thread(null, doBackgroundThreadProcessing,
                               "Background");
    thread.start();
}

// 이 Runnable은 백그라운드에서 수행할 작업을 가진 메서드를 실행한다.
private Runnable doBackgroundThreadProcessing = new Runnable() {
    public void run() {
        backgroundThreadProcessing();
    }
};

// 백그라운드에서 수행할 작업을 가진 메서드.
private void backgroundThreadProcessing() {
    [ ... 시간이 많이 드는 작업 ... ]
}
```

## 핸들러로 GUI 작업 수행하기

GUI 환경에서 백그라운드 스레드를 사용할 때는 그래픽 요소들을 만들거나 변경하기에 앞서 자식 스레드를 메인 애플리케이션(GUI) 스레드와 동기화하는 것이 중요하다.

애플리케이션 컴포넌트에서 알림과 인텐트는 항상 GUI 스레드에서 수신되고 처리된다. 이 외에 GUI 스레드에서 생성된 객체나 토스트처럼 화면에 메시지를 표시하는 객체와 명시적으로 상호작용하는 작업들은 반드시 메인 스레드에서 실행되어야 한다.

액티비티의 경우에는 `runOnUiThread` 메서드를 이용할 수도 있다. `runOnUiThread`는 메서드를 강제로 액티비티 UI와 동일한 스레드에서 실행할 수 있게 해준다.


**코드 9-12** 액티비티의 GUI 스레드와 동기화하기

```
runOnUiThread(new Runnable() {
    public void run() {
        // TODO: 뷰를 업데이트한다.
    }
});
```

(토스트와 알림처럼) 액티비티가 아닌 다른 상황에서는 `Handler` 클래스를 이용해 메서드를 `Handler` 클래스가 생성된 스레드로 보낼 수 있다.

Handler 클래스의 post 메서드를 이용하면 백그라운드 스레드에서 사용자 인터페이스 업데이트 작업을 보낼 수 있다. 코드 9-13은 핸들러를 이용해 GUI 스레드를 업데이트하기 위한 개요를 보여준다.


Available for download on Wrox.com

**코드 9-13** 핸들러를 이용해 GUI 스레드와 동기화하기

```
// 메인 스레드에서 핸들러를 초기화한다.
private Handler handler = new Handler();

private void mainProcessing() {
    Thread thread = new Thread(null, doBackgroundThreadProcessing,
                               "Background");
    thread.start();
}

private Runnable doBackgroundThreadProcessing = new Runnable() {
    public void run() {
        backgroundThreadProcessing();
    }
};

// 백그라운드에서 수행할 작업을 가진 메서드.
private void backgroundThreadProcessing() {
    [ ... 시간이 많이 드는 작업 ... ]
    handler.post(doUpdateGUI);
}

// GUI 업데이트 메서드를 실행하는 Runnable.
private Runnable doUpdateGUI = new Runnable() {
    public void run() {
        updateGUI();
    }
};

private void updateGUI() {
    [ ... 다이얼로그를 열거나 GUI 요소를 변경한다 ... ]
}
```

또한 Handler 클래스의 postDelayed 메서드를 이용하면 GUI 업데이트 작업이 보내지는 시점을 늦출 수 있고, postAtTime 메서드를 이용하면 원하는 시점에 보낼 수 있다.

## 토스트 만들기

토스트는 일시적인 성격의 다이얼로그 박스로서 화면에 몇 초간 보였다가 사라진다. 토스트는 포커스를 뺏지 않으며, 모들이 아니므로 활성화된 애플리케이션을 방해하지 않는다.

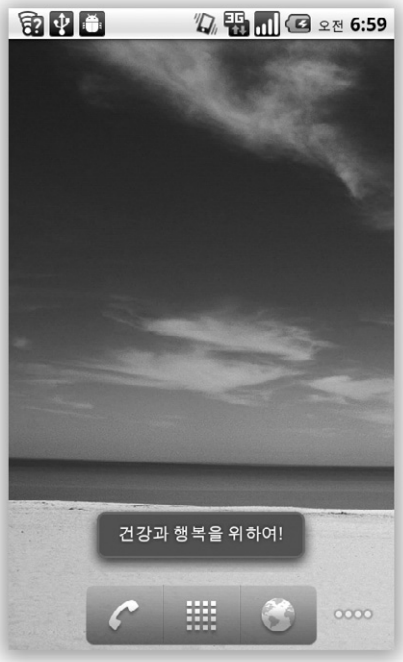



그림 9-1

토스트를 이용하면 사용자에게 특정 액티비티를 열거나 알림을 읽도록 강요하지 않고도 정보를 전달할 수 있다. 토스트는 포그라운드 애플리케이션을 방해하지 않고, 백그라운드 서비스에서 발생하는 이벤트를 사용자에게 알리는 데 적합한 메커니즘을 제공한다.

Toast 클래스에 있는 static 메서드 makeText는 표준 토스트 윈도우를 생성한다. 새로운 토스트를 만들려면 makeText 메서드에 애플리케이션 컨텍스트, 표시할 텍스트 메시지, 그리고 토스트가 화면에 머무는 시간 길이(LENGTH_SHORT나 LENGTH_LONG)를 전달한다. 이렇게 만든 토스트를 화면에 표시하려면 코드 9-14에서 보이는 것처럼 show를 호출한다.

Available for download on Wrox.com

**코드 9-14** 토스트 표시하기

```
Context context = getApplicationContext();
String msg = "건강과 행복을 위하여!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, msg, duration);
toast.show();
```

그림 9-1은 코드 9-14를 이용해 표시한 토스트를 보여준다. 이 토스트는 약 2초간 화면에 머물다 사라질 것이다. 토스트가 화면에 표시되고 있는 동안에도 토스트 뒤에 있는 애플리케이션은 완벽히 반응하며 인터랙티브한 상태로 남는다.

## 토스트 커스터마이즈하기

보통은 텍스트 메시지를 표시하는 표준 토스트 윈도우만으로도 충분하지만, 상황에 따라 토스트의 겉모양과 위치를 커스터마이즈하고 싶을 수도 있다. 이럴 때는 토스트의 표시 위치를 설정하고 대체 뷰나 레이아웃을 할당해 토스트를 수정할 수 있다.

코드 9-15는 `setGravity` 메서드로 토스트를 화면 아래에 정렬하는 법을 보여준다.


Available for download on Wrox.com

**코드 9-15** 토스트 커스터마이즈하기

```
Context context = getApplicationContext();
String msg = "신랑과 신부를 위하여!";
int duration = Toast.LENGTH_SHORT;
Toast toast = Toast.makeText(context, msg, duration);
int offsetX = 0;
int offsetY = 0;

toast.setGravity(Gravity.BOTTOM, offsetX, offsetY);
toast.show();
```

텍스트 메시지만으로 부족한 경우에는 커스텀 뷰나 레이아웃을 지정해 좀더 복잡하고 비주얼한 토스트 윈도우를 구성할 수 있다. 토스트 객체에 `setView`를 이용하면 어떠한 뷰든 지정할 수 있다(레이아웃 포함). 이렇게 지정된 뷰는 표준 토스트 윈도우처럼 일시적인 메시지 윈도우 메커니즘transient message window mechanism을 통해 화면에 머물다 사라진다.

예컨대, 코드 9-16은 4장에서 만든 `CompassView` 위젯과 `TextView`를 가지고 있는 레이아웃을 토스트로 표시하도록 설정한다.


Available for download on Wrox.com

**코드 9-16** 뷰를 이용해 토스트 커스터마이즈하기

```
Context context = getApplicationContext();
String msg = "건배!";
int duration = Toast.LENGTH_LONG;
Toast toast = Toast.makeText(context, msg, duration);
toast.setGravity(Gravity.TOP, 0, 0);

LinearLayout ll = new LinearLayout(context);
ll.setOrientation(LinearLayout.VERTICAL);

TextView myTextView = new TextView(context);
```

```
CompassView cv = new CompassView(context);

myTextView.setText(msg);

int lHeight = LinearLayout.LayoutParams.FILL_PARENT;
int lWidth = LinearLayout.LayoutParams.WRAP_CONTENT;

ll.addView(cv, new LinearLayout.LayoutParams(lHeight, lWidth));
ll.addView(myTextView, new LinearLayout.LayoutParams(lHeight, lWidth));

ll.setPadding(40, 50, 0, 50);

toast.setView(ll);
toast.show();
```

위 코드가 만들어내는 토스트는 그림 9-2처럼 화면에 나타난다.


그림 9-2

## 작업자 스레드에서 토스트 사용하기

토스트는 GUI 컴포넌트로서 반드시 GUI 스레드에서 열려야 한다. 그렇지 않으면 교차 스레드 예외cross thread exception가 발생한다. 코드 9-17은 핸들러를 이용해 토스트가 GUI 스레드에서 열리도록 보장한다.


Available for download on Wrox.com

**코드 9-17** GUI 스레드에서 토스트 띄우기

```
private void mainProcessing() {
    Thread thread = new Thread(null, doBackgroundThreadProcessing,
                               "Background");
    thread.start();
}

private Runnable doBackgroundThreadProcessing = new Runnable() {
    public void run() {
        backgroundThreadProcessing();
    }
};

private void backgroundThreadProcessing() {
    handler.post(doUpdateGUI);
}

// GUI 업데이트 메서드를 실행하는 Runnable.
private Runnable doUpdateGUI = new Runnable() {
    public void run() {
        Context context = getApplicationContext();
        String msg = "열린 모바일 개발을 위하여!";
        int duration = Toast.LENGTH_SHORT;
        Toast.makeText(context, msg, duration).show();
    }
};
```

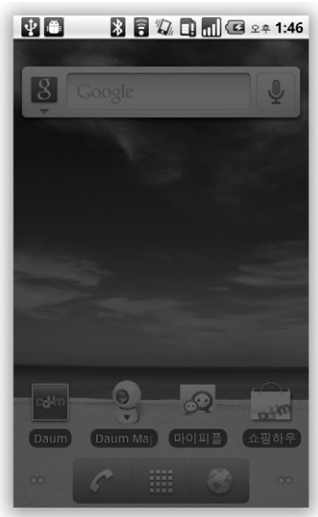
## 알림 소개[6]

알림을 이용하면 액티비티를 쓰지 않고도 사용자에게 정보를 전달할 수 있다. 알림은 알림 매니저를 통해 처리된다. 현재 알림으로 할 수 있는 일은 다음과 같다.

- 새로운 상태 표시줄 아이콘을 만든다.
- 펼쳐진 상태 표시줄 윈도우에 추가 정보를 표시하고 인텐트를 발생시킨다.
- 불빛과 LED를 깜박인다.
- 폰에 진동을 울린다.
- 벨 소리와 미디어 스토어 오디오 등의 가청 경보audible alerts를 울린다.

알림은 화면에 보이지 않는 애플리케이션 컴포넌트들이 사용자에게 이벤트를 알리는 데 선호하는 방법이다. 이러한 애플리케이션 컴포넌트에는 브로드캐스트 리시버, 서비스, 비활성화된 액티비티가 있으며, 주로 사용자의 확인이 필요한 이벤트를 알리는 데 쓰인다. 또한 알림은 실행 중인 백그라운드 서비스를 나타내는데도 쓰인다(특히 포그라운드 우선순위로 설정된 서비스).

⑥ **옮긴이** 원서에서는 알림과 관련된 UI 요소들을 언급하면서 서로 다른 용어를 사용하고 있어 무척 혼란스럽다. 이번 절에서 다룰 알림과 관련된 UI 요소들과 이를 지칭하는 용어들을 정리해보자.



상태 표시줄

펼쳐진 상태 표시줄 윈도우

알림 표시줄

사용자 인터페이스 메타포로서 알림은 특히 모바일 장치에 적합하다. 사용자가 항상 휴대폰을 가지고 다니더라도 휴대폰 화면을 계속 쳐다보고 다닐 수는 없는 노릇이다. 애플리케이션도 마찬가지다. 사용자는 보통 여러 애플리케이션들을 백그라운드에 열어두고 사용하는데, 새로운 정보를 받아보기 위해 이들 애플리케이션을 일일이 살피고 있을 수는 없다.


그림 9-3

이런 환경에서는 애플리케이션의 알림 기능이 중요하다. 사용자의 확인이 필요한 특정 이벤트가 발생하면 애플리케이션이 이를 사용자에게 알려줄 수 있어야 한다.

알림은 계속 지속될 수 있다. 사용자가 취소할 때까지 반복하거나, 진행 중으로 표시하거나, 아니면 그냥 단순히 상태 표시줄에 아이콘을 표시해두면 된다. 상태 표시줄 아이콘은 규칙적으로 업데이트될 수 있으며, 그림 9-3에 보이는 펼쳐진 상태 표시줄 윈도우를 이용해 추가 정보를 보여주도록 확장될 수도 있다.

상태 표시줄 윈도우를 펼치려면 상태 표시줄을 클릭한 상태에서 화면 아래로 드래그한다. 상태 표시줄 윈도우를 완전히 펼치려면 윈도우가 화면 전체를 덮을 때까지 드래그한 뒤 손을 떼면 된다. 상태 표시줄 윈도우를 다시 숨기려면 화면 위로 드래그한다.

## 알림 매니저 소개

알림 매니저Notification Manager는 알림을 다루는 데 사용되는 시스템 서비스다. 알림 매니저의 레퍼런스는 코드 9-18에서 보이는 것처럼 `getSystemService` 메서드로 얻는다.


**코드 9-18** 알림 매니저 이용하기

```
String svcName = Context.NOTIFICATION_SERVICE;
```

```
NotificationManager notificationManager;
notificationManager = (NotificationManager)getSystemService(svcName);
```

알림 매니저를 이용하면 새로운 알림을 생성할 수 있고, 기존 알림을 변경할 수 있으며, 더 이상 필요치 않은 알림을 제거할 수 있다.

## 알림 생성하기

안드로이드가 제공하는 알림을 이용한 정보 전달 방법에는 여러 가지가 있다.

**1.** 상태 표시줄 아이콘

**2.** 펼쳐진 상태 표시줄 윈도우

**3.** 소리와 진동 같은 추가 폰 효과

이번 절에서는 앞에 있는 두 가지를 살펴본다. 이번 장 후반부에서는 Notification 객체의 다양한 속성을 이용해 장치 LED를 깜박이고, 폰에 진동을 울리고, 오디오를 재생하도록 알림을 향상시키는 법을 배운다.

### 알림 생성하기와 상태 표시줄 아이콘 구성하기

먼저 코드 9-19에서 보이는 것처럼 새로운 Notification 객체를 생성하고, 상태 표시줄에 표시할 아이콘과 상태 표시줄 티커 텍스트ticker text 그리고 이 알림의 생성 시간을 전달한다.


**코드 9-19** 알림 생성하기

```
// 상태 표시줄 아이콘으로 쓸 드로어블.
int icon = R.drawable.icon;
// 알림 생성 시 상태 표시줄에 표시할 텍스트.
String tickerText = "알림";
// 펼쳐진 상태 표시줄 윈도우는 알림을 시간 순으로 정렬해 보여준다.
long when = System.currentTimeMillis();

Notification notification = new Notification(icon, tickerText, when);
```

티커 텍스트는 알림 발생 시 상태 표시줄을 따라 스크롤된다.

또한 Notification 객체의 number 속성을 설정하면, 상태 표시줄 아이콘이 나타내는 이벤

트의 개수를 표시할 수도 있다. 이 값을 다음과 같이 1보다 큰 수로 설정하면, 이 값이 상태 표시줄 아이콘 위에 겹쳐 표시된다.

```
notification.number++;
```

Notification을 변경하고 난 뒤에는 알림을 다시 발생시켜야 변경된 내용이 적용된다. 상태 표시줄 아이콘에 겹쳐 표시된 숫자를 없애려면 number에 0이나 -1을 설정한다.

### 알림 표시줄 구성하기

펼쳐진 상태 표시줄 윈도우에 표시되는 알림의 겉모양을 구성하는 방법에는 두 가지가 있다.

1. 표준 알림 표시줄을 이용한다. 표준 알림 표시줄에 표시되는 세부 정보들은 setLatestEventInfo 메서드로 업데이트한다.
2. 커스텀 UI를 이용한다. 알림 표시줄에 커스텀 UI를 적용하려면 리모트 뷰를 생성한 뒤 contentView 속성에 설정한다.

가장 간단한 방법은 setLatestEventInfo 메서드로 표준 알림 표시줄 레이아웃을 채우는 것이다. 표준 알림 표시줄 레이아웃은 그림 9-4에서 보이는 것처럼 아이콘, 알림 생성 시 정의된 시간, 제목, 세부 정보 문자열을 보여준다.

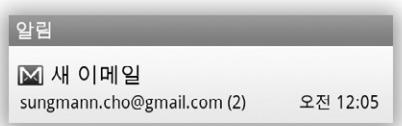



그림 9-4

알림은 액션 요청이나 확인 요청을 나타내기도 한다. 때문에 알림에는 사용자가 알림을 클릭할 때 발생시킬 PendingIntent를 지정할 수 있다. 특별한 경우가 아니라면 이 인텐트는 애플리케이션을 열어 해당 알림의 의도에 맞는 액티비티를 찾아 보여줘야 한다.[7]

코드 9-20은 setLatestEventInfo를 이용해 알림 값을 설정한다.

Available for download on Wrox.com

**코드 9-20** 알림 값 설정하기

```
Context context = getApplicationContext();
// 알림 표시줄에 표시할 텍스트.
```

[7] **옮긴이** 예를 들어, 그림 9-4에 있는 G메일의 "새 이메일" 알림을 클릭하면, G메일 애플리케이션이 열리고 받은 편지함 액티비티가 화면에 나타난다.

```
String expandedText = "알림 표시줄 텍스트";
// 알림 표시줄 제목
String expandedTitle = "알림 제목";
// 알림 표시줄이 클릭될 경우 액티비티를 실행하기 위한 인텐트.
Intent intent = new Intent(this, MyActivity.class);
PendingIntent launchIntent = PendingIntent.getActivity(context, 0, intent, 0);

notification.setLatestEventInfo(context,
                                expandedTitle,
                                expandedText,
                                launchIntent);
```

(여러 개의 SMS 메시지가 수신된 경우처럼) 같은 이벤트의 여러 인스턴스는 하나의 알림 아이콘으로 나타내는 것이 좋은 형태다. 이렇게 하려면 가장 최근 메시지(또는 여러 메시지들의 요약 정보)를 반영해 setLatestEventInfo로 설정된 값을 업데이트하고, 다시 알림을 발생시켜 화면에 표시되는 값을 업데이트한다.

표준 알림 표시줄로 표현할 수 있는 내용이 부족하거나 적합하지 않은 경우에는 여러분이 직접 레이아웃을 만들어 알림에 적용할 수 있다. 커스텀 레이아웃을 알림에 적용할 때는 리모트 뷰를 이용한다. 그림 9-5는 코드 9-21, 9-22, 9-23에서 각각 정의되고 적용되고 수정된 커스텀 레이아웃을 보여준다.

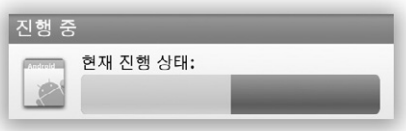



그림 9-5

코드 9-21은 아이콘, 텍스트 뷰, 프로그래스 바를 가진 커스텀 레이아웃을 정의한다.


**코드 9-21** 알림 표시줄을 위한 커스텀 레이아웃 만들기

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:padding="5dp"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <ImageView
        android:id="@+id/status_icon"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_alignParentLeft="true"
    />
```

```
    <RelativeLayout
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:paddingLeft="10px"
        android:layout_toRightOf="@id/status_icon">
        <TextView
            android:id="@+id/status_text"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_alignParentTop="true"
            android:textColor="#000"
            android:textSize="14sp"
            android:textStyle="bold"
        />
        <ProgressBar
            android:id="@+id/status_progress"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_below="@id/status_text"
            android:progressDrawable="@android:drawable/progress_horizontal"
            android:indeterminate="false"
            android:indeterminateOnly="false"
        />
    </RelativeLayout>
</RelativeLayout>
```

알림에 커스텀 레이아웃을 적용하려면 새로운 RemoteView 객체를 만들어 contentView 속성에 설정한다. 또한 contentIntent 속성에 팬딩 인텐트도 설정한다. 코드 9-22는 진행 중 알림ongoing Notification에 커스텀 레이아웃을 적용한다.


Available for download on Wrox.com

**코드 9-22** 알림 표시줄에 커스텀 레이아웃 적용하기

```
Notification notification = new Notification(R.drawable.icon,
                                            "커스텀 콘텐트",
                                            System.currentTimeMillis());
notification.flags = notification.flags | Notification.FLAG_ONGOING_EVENT;

notification.contentView = new RemoteViews(this.getPackageName(),
                                           R.layout.my_status_window_layout);

Intent intent = new Intent(this, MyActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, intent, 0);
notification.contentIntent = pendingIntent;
```

여러분이 직접 contentView 속성을 설정할 때는 반드시 contentIntent도 함께 설정해야 한다. 그렇지 않으면 알림이 발생할 때 예외가 던져지게 된다.

리모트 뷰는 별도의 애플리케이션 안에 레이아웃을 끼워 넣고 제어할 수 있게 해주는 메커니즘으로, 주로 홈 스크린 위젯을 만들 때 쓰인다. 리모트 뷰에 사용할 레이아웃을 만들 때는 엄격한 제약사항이 존재하는데, 이에 대해서는 다음 장에서 보다 자세히 다룬다.

알림 표시줄에 사용된 뷰의 속성과 겉모양을 수정하려면, 코드 9-23에서 보이는 것처럼 리모트 뷰 객체가 가진 set* 메서드를 이용한다. 코드 9-23은 코드 9-21에서 정의한 레이아웃에 쓰인 각 뷰를 수정한다.


**코드 9-23** 알림 표시줄 커스터마이즈하기

```
notification.contentView.setImageViewResource(R.id.status_icon,
                                              R.drawable.icon);
notification.contentView.setTextViewText(R.id.status_text,
                                         "현재 진행 상황:");
notification.contentView.setProgressBar(R.id.status_progress,
                                        100, 50, false);
```

특히 이 기법은 파일을 다운로드하거나 미디어를 재생하는 경우처럼 현재 진행 중인 작업의 진행 상황을 알려주고자 할 때 유용하다. 진행 중 알림에 대해서는 잠시 뒤에 자세히 배운다.

## 알림 발생시키기

알림을 발생시키려면 코드 9-24에서 보이는 것처럼 정수로 된 레퍼런스 ID를 가지고 NotificationManager의 notify 메서드를 호출한다.


**코드 9-24** 알림 발생시키기

```
int notificationRef = 1;
notificationManager.notify(notificationRef, notification);
```

이미 발생시킨 알림을 업데이트하려면, 해당 알림과 동일한 레퍼런스 ID를 가지고 NotificationManager의 notify 메서드를 호출해 알림을 다시 발생시킨다. 이때 notify 메서드에는 동일한 Notification 객체를 전달할 수도 있고 새로운 객체를 생성해 전달할 수도 있다. 레퍼런스 ID가 동일하다면 새로운 Notification 객체는 상태 표시줄 아이콘과

알림 표시줄 세부 정보를 대체하는 데 쓰일 것이다.

알림을 취소할 때도 레퍼런스 ID가 쓰인다. 알림을 취소하려면 다음과 같이 취소할 알림의 레퍼런스 ID를 가지고 NotificationManager의 cancel 메서드를 호출한다.

```
notificationManager.cancel(notificationRef);
```

알림을 취소하면 해당 알림의 상태 표시줄 아이콘이 제거되고, 펼쳐진 상태 표시줄 윈도우에서도 사라진다.

## 지진 정보 모니터에 알림과 토스트 넣기

이어지는 예제에서는 EarthquakeService에 알림 기능을 넣어 새로운 지진 정보가 있을 때마다 알림을 발생시키도록 만든다. 상태 표시줄 아이콘을 보여주며, 알림 표시줄에는 최근 발생한 지진의 진도와 진원지를 보여준다. 또한 사용자가 알림 표시줄을 선택하면 Earthquake 액티비티를 열어 보여준다.

**1.** 먼저 EarthquakeService 안에 새로운 Notification 인스턴스 변수 하나를 만드는 것으로 시작한다. 이 변수는 상태 표시줄 아이콘과 알림 표시줄의 세부 정보를 제어하는 데 쓰이는 Notification 객체를 저장하는 데 사용된다.[8]

```
private Notification newEarthquakeNotification;
public static final int NOTIFICATION_ID = 1;
```

**2.** onCreate 메서드를 확장해 이 Notification 객체를 생성하도록 구현한다.

```
@Override
public void onCreate() {
    updateTimer = new Timer("earthquakeUpdates");

    int icon = R.drawable.icon;
    String tickerText = "새로운 지진 정보 감지";
    long when = System.currentTimeMillis();

    newEarthquakeNotification= new Notification(icon,
```

[8] **옮긴이** 아래의 import 문도 추가한다.

```
import android.app.Notification;
```

```
                                                    tickerText,
                                                    when);
    }
```

3. 이제 EarthquakeLookupTask 구현으로 넘어가자. onProgressUpdate 스텝을 수정해 각각의 새로운 지진 정보가 콘텐트 프로바이더에 추가되고 나면 알림을 발생시키도록 확장한다. 알림을 발생시키기에 앞서 setLatestEventInfo를 이용해 알림 표시줄의 세부 정보를 업데이트한다. 또한 토스트를 만들어 표시해 새로운 지진 정보를 알린다.[9]

```
@Override
protected void onProgressUpdate(Quake... values) {
    String svcName = Context.NOTIFICATION_SERVICE;
    NotificationManager notificationManager =
        (NotificationManager)getSystemService(svcName);
    Context context = getApplicationContext();
    String expandedText = values[0].getDate().toString();
    String expandedTitle = "M:" + values[0].getMagnitude() + " " +
                           values[0].getDetails();
    Intent startActivityIntent = new Intent(EarthquakeService.this,
                                            Earthquake.class);

    PendingIntent launchIntent =
        PendingIntent.getActivity(context, 0, startActivityIntent, 0);

    newEarthquakeNotification.setLatestEventInfo(context,
                                                 expandedTitle,
                                                 expandedText,
                                                 launchIntent);
    newEarthquakeNotification.when =
        java.lang.System.currentTimeMillis();

    notificationManager.notify(NOTIFICATION_ID,
                               newEarthquakeNotification);

    Toast.makeText(context, expandedTitle, Toast.LENGTH_SHORT).show();
}
```

❾ **옮긴이** 아래의 import 문도 추가한다.

```
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.widget.Toast;
```

**4.** 마지막 단계는 Earthquake 액티비티와 EarthquakeMap 액티비티에서 알림을 비우고 비활성화시키는 것이다. 액티비티가 활성화될 때 상태 표시줄 아이콘을 없애면 된다.

**4.1.** Earthquake 액티비티부터 시작하자. 알림 onCreate 메서드를 수정해 알림 매니저의 레퍼런스를 얻어오도록 구현한다.[10]

```
NotificationManager notificationManager;

@Override
public void onCreate(Bundle savedInstanceState) {
    [ ... 기존 onCreate ... ]

    String svcName = Context.NOTIFICATION_SERVICE;
    notificationManager =
        (NotificationManager)getSystemService(svcName);
}
```

**4.2.** EarthquakeReceiver의 onReceive 메서드를 수정한다. 이 브로드캐스트 리시버는 액티비티가 활성화될 때만 등록된다. 즉, onReceive 메서드는 액티비티가 활성화된 경우에만 실행된다는 얘기다. 따라서 onReceive 메서드가 호출되자마자 모든 지진 정보 알림을 취소해도 안전하다.

```
@Override
public void onReceive(Context context, Intent intent) {
    loadQuakesFromProvider();

    notificationManager.cancel(EarthquakeService.NOTIFICATION_ID);
}
```

**4.3.** 다음으로 onResume 메서드를 확장해 액티비티가 활성화되면 알림을 취소하도록 구현한다.

```
@Override
public void onResume() {
    notificationManager.cancel(EarthquakeService.NOTIFICATION_ID);
```

[10] **옮긴이** 아래의 import 문도 추가한다.

```
import android.app.NotificationManager;
```

```
    IntentFilter filter;
    filter = new IntentFilter(EarthquakeService.NEW_EARTHQUAKE_FOUND);
    receiver = new EarthquakeReceiver();
    registerReceiver(receiver, filter);
    super.onResume();
}
```

**4.4.** 지금까지 Earthquake 액티비티에 한 것과 동일한 처리를 EarthquakeMap 액티비티에도 적용한다.[11]

```
NotificationManager notificationManager;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.earthquake_map);

    ContentResolver cr = getContentResolver();
    earthquakeCursor = cr.query(EarthquakeProvider.CONTENT_URI,
                                null, null, null, null);

    MapView earthquakeMap = (MapView)findViewById(R.id.map_view);
    earthquakeMap.getOverlays().add(new EarthquakeOverlay(earthquakeCursor));

    String svcName = Context.NOTIFICATION_SERVICE;
    notificationManager = (NotificationManager)getSystemService(svcName);
}

@Override
public void onResume() {
    notificationManager.cancel(EarthquakeService.NOTIFICATION_ID);

    earthquakeCursor.requery();

    IntentFilter filter;
    filter = new IntentFilter(EarthquakeService.NEW_EARTHQUAKE_FOUND);
    receiver = new EarthquakeReceiver();
    registerReceiver(receiver, filter);
```

[11] **옮긴이** 아래의 import 문도 추가한다.

```
import android.app.NotificationManager;
```

```
        super.onResume();
    }

    public class EarthquakeReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            notificationManager.cancel(EarthquakeService.NOTIFICATION_ID);

            earthquakeCursor.requery();
            MapView earthquakeMap = (MapView)findViewById(R.id.map_view);
            earthquakeMap.invalidate();
        }
    }
```

## 고급 알림 테크닉

이어지는 절에서는 하드웨어를 통해 추가 알림 기능을 제공하는 법을 배운다. 알림을 발생시킬 때 장치의 벨을 울리거나, 불빛을 깜박이거나, 진동을 울리면 알림의 효과를 좀더 높일 수 있다.

각각의 고급 기능을 설명해나가면서 여러분에게 코드를 제시할 것이다. 코드는 지진 정보 예제에 추가할 수 있는 형태로 주어진다. 이 코드를 지진 정보 예제에 적용하면 감지된 각 지진의 피해 정도에 따라 사용자에게 피드백을 제공할 수 있다.

여기에 설명된 알림 테크닉을 이용할 때 상태 표시줄 아이콘을 표시하지 않으려면, 알림을 발생시킨 다음 간단히 해당 알림을 취소하면 된다. 이렇게 하면 상태 표시줄 아이콘은 표시되지 않지만 기타 다른 효과들은 중단되지 않는다.

### 기본 값 사용하기

알림에 소리, 불빛, 진동을 추가하는 가장 간단하면서도 무난한 방법은 사용자가 설정해둔 기본 값을 사용하는 것이다. Notification의 defaults 속성에는 다음의 값들을 조합해 넣을 수 있다.

- Notification.DEFAULT_LIGHTS
- Notification.DEFAULT_SOUND

■ Notification.DEFAULT_VIBRATE

다음은 Notification에 기본 소리 설정과 기본 진동 설정을 설정하는 코드다.

```
notification.defaults = Notification.DEFAULT_SOUND |
                        Notification.DEFAULT_VIBRATE;
```

모두에 대해 기본 값을 사용하고자 하는 경우에는 Notification.DEFAULT_ALL 상수를 이용할 수 있다.

### 소리내기

사용자에게 수신 전화 같은 장치 이벤트를 소리로 알리는 방법은 모바일 이전에도 있던 것으로 오랜 시간 사용되어 왔다. 수신 전화에서부터 새 메시지와 배터리 부족에 이르기까지 폰에서 발생하는 대부분의 네이티브 이벤트들은 벨 소리를 통해 알 수 있다.

안드로이드에서는 폰에 있는 오디오 파일을 알림으로 사용할 수 있다. 다음의 코드에서 보이는 것처럼 원하는 오디오 파일의 URI를 sound 속성에 설정하면 된다.

```
notification.sound = ringURI;
```

여러분이 가진 커스텀 오디오를 알림으로 사용하려면 해당 파일을 장치에 넣거나 원시 리소스로 포함시킨다. 이에 대해서는 11장에서 설명한다.

이어지는 코드는 이전 예제의 지진 정보 서비스에 있는 announceNewQuake 메서드에 추가할 수 있는 코드로서 지진 정보 알림에 오디오 컴포넌트를 추가하고, 진도가 6 이상인 큰 지진이 발생한 경우 기본 알림 벨 소리를 울린다.

```
if (quake.getMagnitude() > 6) {
    Uri ringURI =
        RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
    newEarthquakeNotification.sound = ringURI;
}
```

### 진동 울리기

안드로이드는 폰의 진동 기능을 이용해 알림별로 진동 패턴vibration pattern을 실행하고, 이 진

동 패턴을 제어할 수 있게 해준다. 진동을 이용하면 사용자의 주의를 끌 수 있을 뿐 아니라 정보를 전달할 수도 있다.

진동 패턴을 설정하려면 일련의 long 값들로 구성된 배열을 Notification의 vibrate 속성에 설정한다. 이 배열을 구성하고 있는 값들은 진동을 울릴 시간 길이와 울리지 않을 시간 길이를 교대로 반복해 나타낸다. 이때 시간 길이는 밀리 초 단위다.

애플리케이션에서 진동 기능을 이용할 수 있으려면 먼저 권한을 부여받아야 한다. 다음의 코드를 이용해 장치의 진동 기능 접근을 요청하는 uses-permission을 애플리케이션 매니페스트에 추가하자.

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

이어지는 예제는 알림에 진동 패턴을 설정하는 법을 보여준다. 이 진동 패턴은 1초 간격으로 진동을 켜고 끄며, 총 5초간 반복한다.

```
long[] vibrate = new long[] { 1000, 1000, 1000, 1000, 1000 };
notification.vibrate = vibrate;
```

진동 패턴을 섬세하게 잘 구성하면 진동만으로도 사용자에게 정보를 전달할 수 있다. 이어지는 코드를 announceNewQuake 메서드에 추가하자. 이 코드는 폰이 지진 규모에 기반한 패턴으로 진동을 울리도록 설정한다. 지진은 지수 크기exponential scale로 측정되므로 진동 패턴 생성 시 이 크기를 그대로 이용한다.

겨우 인지 가능한 진도 1 규모의 지진의 경우 폰은 1초도 진동하지 않을 것이다. 하지만 진도 10 규모의 지진은 지구를 반으로 가를 것이며, 장치가 20초를 꽉 채워 진동할 경우 사용자는 이 대참사를 피해 어디라도 가려 할 것이다. 대부분 의미 있는 지진은 리히터 규모Richter scale 3과 7 사이에 떨어지므로 200 밀리 초에서 4초 범위의 진동 길이가 적당하다.

```
double vibrateLength = 100*Math.exp(0.53*quake.getMagnitude());
long[] vibrate = new long[] {100, 100, (long)vibrateLength };
newEarthquakeNotification.vibrate = vibrate;
```

현재 안드로이드 에뮬레이터는 시각적으로나 청각적으로 장치의 진동 여부를 나타내지 않는다.

### 불빛 깜박이기

Notification에는 장치의 LED 색상과 깜박임 주기를 설정하기 위한 속성 역시 포함되어 있다.

LED 제어와 관련된 제약사항은 장치마다 서로 다를 수 있다. 장치가 여러분이 지정한 색상을 표현할 수 없는 경우에는 가능한 한 근사치에 가까운 색상이 사용될 것이다. LED로 사용자에게 정보를 전달할 때는 이러한 제약사항을 염두에 두자. 또한 정보를 전달할 때는 한 가지 방법만을 고수하지 말고 이러한 제약사항에 유연하게 대처할 수 있도록 준비하자.

ledARGB 속성은 LED의 색상을 설정하는 데 쓰이며, ledOffMS와 ledOnMS 속성은 LED의 깜박임 주기와 패턴을 설정하는 데 쓰인다. ledOnMS 속성을 1로, ledOffMS 속성을 0으로 설정하면 LED가 켜지며, 두 속성 모두 0으로 설정하면 LED가 꺼진다.

LED 설정을 모두 마치고 난 뒤에는 반드시 Notification의 flags 속성에 FLAG_SHOW_LIGHTS 플래그를 추가해야 한다.

다음 코드는 빨간색 장치 LED를 켜는 법을 보여준다.

```
notification.ledARGB = Color.RED;
notification.ledOffMS = 0;
notification.ledOnMS = 1;
notification.flags = notification.flags | Notification.FLAG_SHOW_LIGHTS;
```

색상과 깜박임 주기 역시 잘 제어하면 사용자에게 추가 정보를 제공할 수 있다.

지진 정보 모니터링 예제의 경우, 장치 LED를 이용해 지진의 진도를 표현하면 사용자가 지수 크기의 뉘앙스를 인지하는 데 도움이 될 수 있다. 다음 코드에서 LED 색상은 지진의 규모에 따라 결정되며, 깜박임 주기는 지진의 강도와 반비례 관계로 설정된다.

```
int color;
if (quake.getMagnitude() < 5.4)
    color = Color.GREEN;
else if (quake.getMagnitude() < 6)
    color = Color.YELLOW;
else
    color = Color.RED;
```

```
newEarthquakeNotification.ledARGB = color;
newEarthquakeNotification.ledOffMS = (int)vibrateLength;
newEarthquakeNotification.ledOnMS = (int)vibrateLength;
newEarthquakeNotification.flags = newEarthquakeNotification.flags |
                                  Notification.FLAG_SHOW_LIGHTS;
```

현재 안드로이드 에뮬레이터는 LED를 시각적으로 표현하지 않는다.[12]

## 진행 중 알림과 강조 알림

Notification에 FLAG_ONGOING_EVENT 플래그를 설정하면 진행 중 알림ongoing notifications을, FLAG_INSISTENT 플래그를 설정하면 강조 알림insistent notifications을 만들 수 있다.

다음 코드와 같이 진행 중 플래그가 설정된 알림은 백그라운드에서 파일을 다운로드하거나 음악을 재생하고 있는 경우처럼 현재 진행 중인 이벤트를 나타내는 데 쓰인다. 이번 장 앞에서 설명한 것처럼 포그라운드 서비스의 경우에는 반드시 진행 중 알림을 설정해야 한다.

```
notification.flags = notification.flags |
   Notification.FLAG_ONGOING_EVENT;
```

그림 9-6에서 보이는 것처럼 진행 중 알림은 펼쳐진 상태 표시줄 윈도우에서 보통의 알림과 구분되어 표시된다.

강조 알림은 취소될 때까지 자신에게 설정된 동작(소리를 내거나, 진동을 울리거나, 불빛을 깜박이는 등의 동작)을 계속 반복한다. 보통 강조 알림은 전


그림 9-6

⑫ **옮긴이** LED 관련 기능을 넥서스 원(Nexus One)으로 테스트해본 결과, 화면 LED가 아닌 트랙볼 색상이 깜박임 주기대로 변하는 것을 확인했다. 이처럼 LED 기능은 하드웨어에 따라 결과가 다르게 나타난다는 사실을 꼭 염두에 두자.

화가 오거나 알람 시계의 벨이 울리는 것과 같이 신속하고 즉각적인 반응을 필요로 하는 이벤트에 쓰인다.

다음 코드는 강조 알림으로 설정하는 법을 보여준다.

```
notification.flags = notification.flags |
                     Notification.FLAG_INSISTENT;
```

## 알람 이용하기

알람은 미리 정해둔 시간과 간격에 따라 인텐트를 발생시키는 기능으로서 애플리케이션과 독립된 형태로 동작한다.

알람은 애플리케이션 범위 밖에서 설정된다. 때문에 알람은 애플리케이션이 종료된 후에도 애플리케이션 이벤트나 액션을 만드는 데 쓰일 수 있다. 알람은 특히 브로드캐스트 리시버와 함께 사용할 때 더욱 강력한 힘을 발휘한다. 이렇게 하면 애플리케이션이 실행 중인 상태가 아니라도 인텐트를 방송하거나, 서비스를 시작하거나, 심지어 액티비티를 열도록 알람을 설정할 수 있다.

때문에 알람은 애플리케이션의 리소스 요구사항을 줄이는 데 매우 효과적이다. 특히 애플리케이션이 백그라운드로 진입할 때, 서비스와 타이머는 중지하고 알람으로 예약된 액션은 유지하여 애플리케이션의 리소스 요구사항을 크게 줄일 수 있다.

알람은 예컨대 알람 시계 애플리케이션을 구현하거나, 규칙적인 네트워크 조회를 수행하거나, 시간 또는 비용이 많이 드는 작업을 "한산할 때" 수행하도록 계획할 때 사용할 수 있다.

애플리케이션이 살아있는 동안에만 발생하는 타이밍 작업의 경우에는 알람을 이용하기보다 타이머와 스레드를 결합한 Handler 클래스를 이용하는 편이 훨씬 더 낫다. 이렇게 하면 안드로이드가 시스템 리소스를 더 잘 제어할 수 있게 된다. 알람은 예정된 이벤트들을 애플리케이션 제어 밖으로 옮겨 해당 애플리케이션이 실행되어 떠있는 시간을 줄여주는 메커니즘을 제공한다.

안드로이드의 알람은 장치가 절전 상태에 있는 동안에도 활성화된 상태로 남아 있으며, 옵션으로 장치를 깨우도록 설정될 수 있다. 하지만 모든 알람은 장치가 재부팅 될 때마다 취소된다.

알람 작업은 AlarmManager를 통해 처리된다. AlarmManager는 시스템 서비스로서 getSystemService를 통해 접근할 수 있다.

```
AlarmManager alarms = (AlarmManager)getSystemService(Context.ALARM_SERVICE);
```

새로운 일회성 알람one-shot Alarm을 만들려면 set 메서드를 이용해 알람의 종류, 발생 예정 시간, 그리고 알람 생성 시 발생시킬 팬딩 인텐트를 설정한다. 지정된 발생 예정 시간이 현재 시간보다 앞선 경우(즉, 과거로 설정된 경우)에는 알람이 즉시 발생된다.

알람의 종류에는 네 가지가 있다. set 메서드에 전달된 발생 예정 시간은 알람의 종류에 따라 특정 시간으로 해석되기도 하고 경과 대기 시간으로 해석되기도 한다.[18]

- RTC_WAKEUP 지정된 시간에 팬딩 인텐트를 발생시키기 위해 장치를 절전 상태에서 깨운다.
- RTC 지정된 시간에 팬딩 인텐트를 발생시키나 장치를 깨우지는 않는다.
- ELAPSED_REALTIME 장치가 부팅된 이후로 경과된 시간의 양에 기반해 인텐트를 발생시키나 장치를 깨우지는 않는다. 경과 시간에는 장치가 절전 상태로 있었던 시간이 포함된다. 이 경과 시간은 장치가 마지막으로 부팅된 시점을 기준으로 계산된다는 사실을 염두에 두자.
- ELAPSED_REALTIME_WAKEUP 장치가 부팅된 이후 지정된 길이의 시간이 경과하고 나면 장치를 절전 상태에서 깨우고 팬딩 인텐트를 발생시킨다.

코드 9-25는 알람을 만드는 과정을 보여준다.


**코드 9-25** 알람 만들기

```
int alarmType = AlarmManager.ELAPSED_REALTIME_WAKEUP;
long timeOrLengthofWait = 10000;
String ALARM_ACTION = "ALARM_ACTION";
Intent intentToFire = new Intent(ALARM_ACTION);
PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 0,
                                                         intentToFire, 0);

alarms.set(alarmType, timeOrLengthofWait, pendingIntent);
```

[18] **옮긴이** 장치가 절전 상태에 있을 때 알람이 발생할 경우, RTC와 ELAPSED_REALTIME로 설정된 알람은 장치가 깰 때까지 전달되지 않는다.

알람이 발생하면 지정된 팬딩 인텐트가 방송된다. 동일한 팬딩 인텐트로 두 번째 알람을 설정하면 기존 알람을 대체하게 된다.

알람을 취소하려면 다음 코드에서 보이는 것처럼 더 이상 발생되길 원치 않는 팬딩 인텐트를 가지고 알람 매니저에 cancel을 호출한다.

```
alarms.cancel(pendingIntent);
```

코드 9-26에서는 두 개의 알람을 설정하고 이어서 첫 번째 알람을 취소한다. 첫 번째 알람은 특정 시간을 명시적으로 설정하며, 팬딩 인텐트를 발생시키기 위해 장치를 깨운다. 두 번째 알람은 장치가 부팅된 이후 30분이 경과하면 발생되도록 설정되나, 장치를 절전 상태에서 깨우지는 않는다.


**코드 9-26** 알람 설정하고 취소하기

```
AlarmManager alarms =
    (AlarmManager)getSystemService(Context.ALARM_SERVICE);

String MY_RTC_ALARM = "MY_RTC_ALARM";
String ALARM_ACTION = "MY_ELAPSED_ALARM";

PendingIntent rtcIntent =
    PendingIntent.getBroadcast(this, 0,
                               new Intent(MY_RTC_ALARM), 1);
PendingIntent elapsedIntent =
    PendingIntent.getBroadcast(this, 0,
                               new Intent(ALARM_ACTION), 1);

// 5시간 뒤에 장치를 깨우고 팬딩 인텐트를 발생시킨다.
Date t = new Date();
t.setTime(java.lang.System.currentTimeMillis() + 60*1000*5);
alarms.set(AlarmManager.RTC_WAKEUP, t.getTime(), rtcIntent);

// 장치가 부팅된 이후 30분이 경과하면 팬딩 인텐트를 발생시킨다.
alarms.set(AlarmManager.ELAPSED_REALTIME, 30*60*1000, elapsedIntent);

// 첫 번째 알람을 취소한다.
alarms.cancel(rtcIntent);
```

## 반복 알람 설정하기

알람 매니저는 규칙적으로 예약된 이벤트가 필요한 상황을 위해 반복 알람을 설정할 수 있게 해준다. 반복 알람은 앞에서 설명한 일회성 알람과 정확히 동일한 방식으로 동작하지만, 취소될 때까지 지정된 간격을 두고 계속 발생된다는 점이 다르다.

알람은 애플리케이션 컨텍스트 밖에서 설정되기 때문에 백그라운드에서 계속 실행되는 서비스 없이 규칙적인 업데이트나 데이터 조회를 계획하는 데 적합하다.

반복 알람을 설정하려면 코드 9-27에서 보이는 것처럼 알람 매니저의 setRepeating이나 setInexactRepeating 메서드를 이용한다. 두 메서드 모두 앞 절에서 설명한 알람의 종류, 발생 예정 시간, 그리고 알람 생성 시 발생시킬 팬딩 인텐트를 지원한다.

반복 알림의 간격을 정확하고 세밀하게 제어하고 싶을 때는 setRepeating 메서드를 이용하자. 이 메서드를 이용하면 알람의 간격을 밀리 초 단위로 정확히 지정할 수 있다.

setInexactRepeating 메서드는 규칙적으로 업데이트를 수행하도록 스케줄링된 장치를 깨우는 데 드는 배터리 소모를 줄이기 위한 강력한 기법이다. 이 메서드는 정확한 간격 대신 AlarmManager에 정의된 다음의 상수들 중 하나를 받아들인다.

- INTERVAL_FIFTEEN_MINUTES
- INTERVAL_HALF_HOUR
- INTERVAL_HOUR
- INTERVAL_HALF_DAY
- INTERVAL_DAY

안드로이드는 실행 시 복수의 부정확한 반복 알람inexact repeating alarms을 동기화해 일제히 발생시킨다. 이는 각 애플리케이션이 장치를 비슷한 시간대에 개별적으로 깨우는 것을 막는다. 이들 알람을 동기화함으로써 시스템은 규칙적인 반복 이벤트가 배터리에 미치는 영향을 제한할 수 있다.


Available for download on Wrox.com

**코드 9-27** 반복 알람 설정하기

```
// 장치가 깨어있는 경우, 정확히 한 시간마다 팬딩 인텐트를 발생시킨다.
alarms.setRepeating(AlarmManager.ELAPSED_REALTIME_WAKEUP,
```

```
                    60*60*1000, 60*60*1000, elapsedIntent);

// 대략 한 시간마다 장치를 깨우고 알람을 발생시킨다.
alarms.setInexactRepeating(AlarmManager.ELAPSED_REALTIME_WAKEUP,
                           60*60*1000, AlarmManager.INTERVAL_DAY,
                           elapsedIntent);
```

규칙적인 반복 알람이 배터리에 미치는 영향은 상당히 클 수 있다. 반복 알람을 이용할 때는 되도록 이면 느린 주기를 선택해 사용하고, 필요할 때만 장치를 깨우며, 가능한 한 부정확한 반복 알람을 사용하는 것이 좋은 습관이다.

## 반복 알람으로 지진 정보 업데이트하기

이번 예제는 지진 정보 애플리케이션의 마지막 수정으로 지진 정보의 네트워크 업데이트를 예약하는 데 사용 중인 타이머를 알람을 교체한다.

이 방식의 최대 장점 중 하나는 업데이트 완료 시 서비스가 스스로 종료하게 하여 서비스가 사용하고 있던 시스템 리소스를 해제할 수 있게 해준다는 것이다.

**1.** 먼저 BroadcastReceiver를 확장하는 EarthquakeAlarmReceiver라는 이름의 새로운 클래스를 만든다.

```
package com.paad.earthquake;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class EarthquakeAlarmReceiver extends BroadcastReceiver {
}
```

**2.** onReceive 메서드를 재정의한 뒤 EarthquakeService를 명시적으로 시작하도록 구현한다.

```
@Override
public void onReceive(Context context, Intent intent) {
    Intent startIntent = new Intent(context, EarthquakeService.class);
    context.startService(startIntent);
}
```

**3.** 이 브로드캐스트 리시버를 호출하는 데 사용할 액션을 public static 문자열로 정의한다.

```
public static final String ACTION_REFRESH_EARTHQUAKE_ALARM =
    "com.paad.earthquake.ACTION_REFRESH_EARTHQUAKE_ALARM";
```

**4.** EarthquakeAlarmReceiver를 애플리케이션 매니페스트에 추가하고, 단계 3에서 정의한 액션에 귀 기울이는 <intent-filter> 태그를 포함시킨다.

```
<receiver android:name=".EarthquakeAlarmReceiver">
    <intent-filter>
        <action
            android:name="com.paad.earthquake.ACTION_REFRESH_EARTHQUAKE_ALARM"
        />
    </intent-filter>
</receiver>
```

**5.** EarthquakeService의 onCreate 메서드를 수정해 AlarmManager의 레퍼런스를 얻어오고, 알람 생성 시 발생시킬 PendingIntent를 만들도록 업데이트한다. 기존에 있던 timerTask 초기화 코드는 삭제한다.[14]

```
AlarmManager alarms;
PendingIntent alarmIntent;

@Override
public void onCreate() {
    int icon = R.drawable.icon;
    String tickerText = "New Earthquake Detected";
    long when = System.currentTimeMillis();

    newEarthquakeNotification =
        new Notification(icon, tickerText, when);

    alarms = (AlarmManager)getSystemService(Context.ALARM_SERVICE);

    String ALARM_ACTION;
    ALARM_ACTION =
        EarthquakeAlarmReceiver.ACTION_REFRESH_EARTHQUAKE_ALARM;
    Intent intentToFire = new Intent(ALARM_ACTION);
```

[14] **옮긴이** 아래의 import 문도 추가한다.

```
import android.app.AlarmManager;
```

```
        alarmIntent =
            PendingIntent.getBroadcast(this, 0, intentToFire, 0);
    }
```

**6.** onStartCommand 메서드를 수정해 자동 업데이트가 활성화되어 있는 경우 타이머가 아닌 반복 알람을 이용해 업데이트를 예약하도록 설정한다. 동일한 액션을 가지고 새로운 팬딩 인텐트를 설정할 경우 이전에 있던 모든 알림은 자동 취소된다.

이어서 Service.START_STICKY 대신 Service.START_NOT_STICKY를 리턴하도록 수정한다. 단계 7에서는 서비스가 백그라운드 업데이트를 마친 뒤 중지하도록 구현할 것이다. 알람을 이용하면 지정된 업데이트 주기에 따라 또 다른 업데이트를 발생시킬 수 있다. 따라서 서비스가 업데이트 도중에 종료되더라도 시스템은 서비스를 재시작할 필요가 없다.[15]

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    // 공유 환경설정을 얻어온다.
    Context context = getApplicationContext();
    SharedPreferences prefs =
        PreferenceManager.getDefaultSharedPreferences(context);

    int minMagIndex = prefs.getInt(Preferences.PREF_MIN_MAG, 0);
    if (minMagIndex < 0)
        minMagIndex = 0;

    int freqIndex = prefs.getInt(Preferences.PREF_UPDATE_FREQ, 0);
    if (freqIndex < 0)
        freqIndex = 0;

    boolean autoUpdate =
        prefs.getBoolean(Preferences.PREF_AUTO_UPDATE, false);

    Resources r = getResources();
    int[] minMagValues = r.getIntArray(R.array.magnitude);
    int[] freqValues = r.getIntArray(R.array.update_freq_values);

    minimumMagnitude = minMagValues[minMagIndex];
    int updateFreq = freqValues[freqIndex];
```

[15] **옮긴이** 아래의 import 문도 추가한다.

```
import android.os.SystemClock;
```

```
        if (autoUpdate) {
            int alarmType = AlarmManager.ELAPSED_REALTIME_WAKEUP;
            long timeToRefresh = SystemClock.elapsedRealtime() +
                                 updateFreq*60*1000;
            alarms.setRepeating(alarmType, timeToRefresh,
                                updateFreq*60*1000, alarmIntent);
        }
        else
            alarms.cancel(alarmIntent);

        refreshEarthquakes();

        return Service.START_NOT_STICKY;
    };
```

**7.** EarthquakeLookupTask에 있는 onPostExecute 스텁을 수정해 백그라운드 업데이트가 완료될 경우 stopSelf를 호출하도록 구현한다.

```
@Override
protected void onPostExecute(Void result) {
    stopSelf();
}
```

**8.** updateTimer 인스턴스 변수와 TimerTask의 인스턴스인 doRefresh를 제거한다.

## 요약

백그라운드 서비스는 안드로이드 플랫폼이 가진 매력적인 기능 가운데 하나다. 하지만 애플리케이션에서 백그라운드 서비스를 사용하려면 여러 가지 복잡함이 뒤따른다. 이번 장에서는 화면에 보이지 않는 애플리케이션 컴포넌트들을 이용해 애플리케이션이 백그라운드에서 실행되고 있는 동안에도 처리를 계속 이어나가는 방법을 배웠다.

토스트에 대해서도 살펴봤다. 토스트는 애플리케이션의 포커스를 빼앗거나 작업흐름을 방해하지 않고 사용자에게 정보를 표시할 수 있게 해주는 일시적인 성격의 메시지 박스다.

서비스와 액티비티에서 알람 매니저를 이용해 사용자에게 정보를 보내는 법도 배웠다. 알람 매니저를 이용하면 커스터마이즈된 LED와 진동 패턴 그리고 오디오 파일을 이용해 이벤트 정보를 상세히 전달할 수 있다. 또한 진행 중 알림을 만드는 법과 펼쳐진 상태 표시줄 윈도우

에 나타나는 알림 표시줄의 레이아웃을 커스터마이즈하는 법도 배웠다.

여러분은 알람을 이용해 인텐트를 방송하거나 서비스를 시작하기 위한 일회성 이벤트 및 반복 이벤트를 예약할 수 있었다.

또한 이번 장에서는 다음과 같은 것들을 설명했다.

- 서비스와 액티비티를 바인드해 인텐트 엑스트라보다 좀더 세부적이고 구조화된 인터페이스를 이용하는 법
- 네트워크 조회처럼 시간이 많이 드는 처리를 AsyncTask를 이용해 작업자 스레드로 옮겨 애플리케이션이 좋은 반응성을 유지하게 하는 법
- 비주얼 컨트롤과 토스트를 이용한 작업 수행 시 핸들러를 이용해 자식 스레드와 메인 애플리케이션 GUI를 동기화하는 법

10장에서는 애플리케이션을 홈 스크린에 통합하는 법을 배운다. 다이나믹하고 인터랙티브한 홈 스크린 위젯을 만드는 것으로 시작해 라이브 폴더와 라이브 월 페이퍼를 만드는 것으로 이어간다. 마지막으로 퀵 서치 박스를 소개하며, 애플리케이션의 검색 결과를 홈 스크린 검색 위젯에 띄우는 법을 배운다.