Android Developer Fundamentals V2

Services

Lesson 8

What is a service?

A `Service` is an application component that can perform long-running operations in the background and does not provide a user interface.

What are services good for?

- Network transactions.
- Play music.
- Perform file I/O.
- Interact with a database.

Characteristics of services

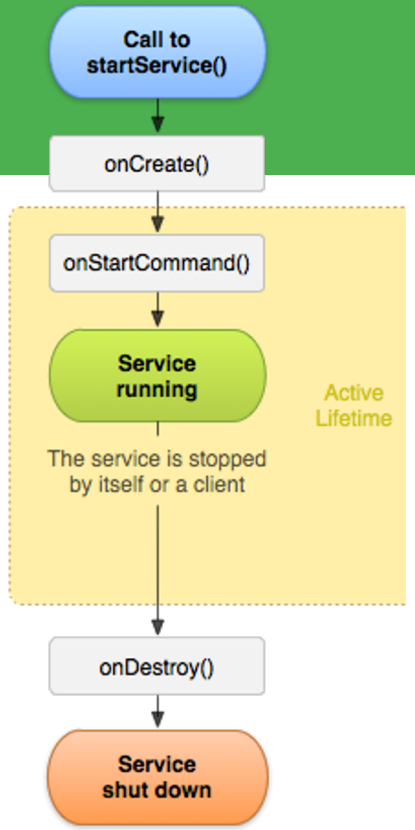
- Started with an `Intent`.
- Can stay running when user switches applications.
- Lifecycle—which you must manage.
- Other apps can use the service—manage permissions.
- Runs in the main thread of its hosting process.

Types of Services

1. **Background Service** (Started Service): a type of service in Android that runs continuously in the background, even when the app is not in the foreground or when the device is in sleep mode
2. **Bound Service**: a type of service in Android that allows other components, such as activities, to bind to it and communicate with it
3. **Foreground Service**: a type of service in Android that has a high priority and is designed to run continuously, even when the app is killed.

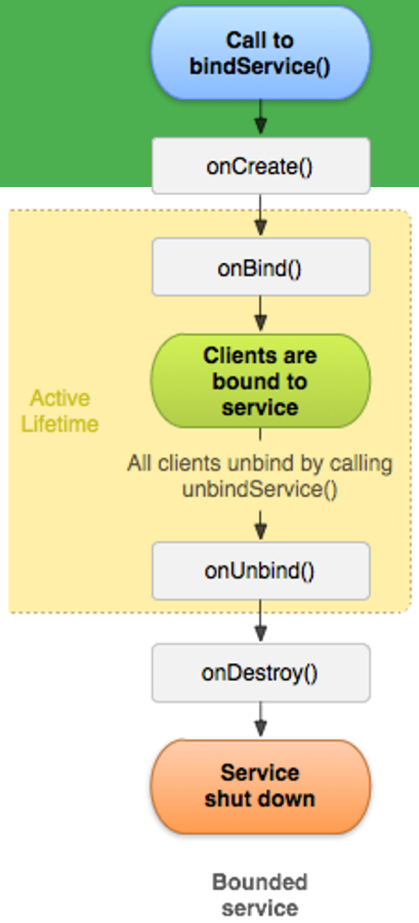
Forms of services: started

- Started with `startService()`
- Runs indefinitely until it stops itself
- Usually does not update the UI



Forms of services: bound

- Offers a client-server interface that allows components to interact with the service
- Clients send requests and get results
- Started with `bindService()`
- Ends when all clients unbind



Services and threads

Although services are separate from the UI, they still run on the main thread by default (except `IntentService`)

Offload CPU-intensive work to a separate thread within the service

Foreground services

Runs in the background but requires that the user is actively aware it exists—e.g. music player using music service

- Higher priority than background services since user will notice its absence—unlikely to be killed by the system
- Must provide a notification which the user cannot dismiss while the service is running

Background services limitations

- Starting from API 26, background app is not allowed to create a background service.
- The `startService()` method now throws an `IllegalStateException` if an app is targeting API 26.
- These limitations don't affect foreground services or bound services.

Creating a service

- `<service android:name=".ExampleService" />`
- Manage permissions.
- Subclass `IntentService` or `Service` class.
- Implement lifecycle methods.
- Start service from `Activity`.
- Make sure service is stoppable.

Stopping a service

- A **started service** must manage its own lifecycle
- If not stopped, will keep running and consuming resources
- The service must stop itself by calling stopSelf()
- Another component can stop it by calling stopService()
- **Bound service** is destroyed when all clients unbound
- **IntentService** is destroyed after onHandleIntent() returns

Started Service Implementation

```
public class ServiceHandler extends Handler {
    @Override
    public void handleMessage(@NonNull Message msg) {
        try {
            while (true) {
                Log.d("handleMessage", "Service is running");
                Thread.sleep(5000);
            }
        } catch (InterruptedException e) {
            // Restore interrupt status.
            Thread.currentThread().interrupt();
        }
    }
}
```

Started Service Implementation

```
public class MyService extends Service {
    private Looper serviceLooper;
    private ServiceHandler serviceHandler;

    @Override
    public void onCreate() {
        HandlerThread thread = new HandlerThread("ServiceStartArguments",
                                                 Process.THREAD_PRIORITY_BACKGROUND);
        thread.start();

        serviceLooper = thread.getLooper();
        serviceHandler = new ServiceHandler(serviceLooper);
    }
}
```

Started Service Implementation

```
public class MyService extends Service {
    private Looper serviceLooper;
    private ServiceHandler serviceHandler;
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.d("MyService", "service starting");
        Message msg = serviceHandler.obtainMessage();
        serviceHandler.sendMessage(msg);


        return START_STICKY;
    }
}
```

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

Bound Service Implementation

```
public class ServiceHandler extends Handler {
    @Override
    public void handleMessage(@NonNull Message msg) {
        try {
            while (true) {
                Log.d("handleMessage", "Service is running");
                Thread.sleep(5000);
            }
        } catch (InterruptedException e) {
            // Restore interrupt status.
            Thread.currentThread().interrupt();
        }
    }
}
```

Bound Service Implementation

```
public class MyService extends Service {
    private Looper serviceLooper;
    private ServiceHandler serviceHandler;

    @Override
    public void onCreate() {
        HandlerThread thread = new HandlerThread("ServiceStartArguments",
                                                 Process.THREAD_PRIORITY_BACKGROUND);
        thread.start();

        serviceLooper = thread.getLooper();
        serviceHandler = new ServiceHandler(serviceLooper);
    }
}
```

Bound Service Implementation

```
public class MyService extends Service {
    private Looper serviceLooper;
    private ServiceHandler serviceHandler;
    private IBinder serviceBinder = new ServiceBinder();

    public IBinder onBind(Intent intent) {
        return serviceBinder;
    }
    public class ServiceBinder extends Binder {
        public MyService getService() {
            return MyService.this;
        }
    }
}
```

Bound Service Implementation

```
ServiceConnection serviceConnection = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName name, IBinder binder) {
        Log.d("ServiceConnection", "onServiceConnected");
        MyService myService = ((MyService.ServiceBinder) binder).getService();
        myService.run();
    }

    @Override
    public void onServiceDisconnected(ComponentName name) {
        Log.d("ServiceConnection", "onServiceDisconnected");
    }
};
```

```
Intent intent = new Intent(MainActivity.this, MyService.class);
bindService(intent, serviceConnection, BIND_AUTO_CREATE);
```

IntentService

IntentService

- Simple service with simplified lifecycle
- Uses worker threads to fulfill requests
- It simplifies the process of managing a worker thread and the Service lifecycle, since it automatically handles the creation, start, and stop of the Service for you
- Ideal for one long task on a single background thread

IntentService restrictions

- `IntentService` are subjected to the new restrictions on background services.
- For the apps targeting API 26, Android Support Library 26.0.0 introduces a new `JobIntentService`.
- `JobIntentService` provides the same functionality as `IntentService` but uses jobs instead of services.

IntentService Implementation

```
public class DownloadService extends IntentService {

    public DownloadService() {
        super("DownloadService");
    }

    @Override
    protected void onHandleIntent(Intent intent) {
        String fileUrl = intent.getStringExtra("fileUrl");
        // Download the file here
        // ...
    }
}
```

```
Intent intent = new Intent(this, DownloadService.class);
intent.putExtra("fileUrl", "https://example.com/file.txt");
startService(intent);
```

JobIntentService

- A subclass of Service in Android that provides a simple way to perform background work asynchronously
- It was introduced in Android 5.0 (API level 21) as an improvement over the IntentService class.
- It supports batching of multiple intents, it can process several intents at once, and provides automatic retries if the work fails due to errors or interruptions.

JobIntentService Implementation

```
public class DownloadService extends JobIntentService {

    private static final int JOB_ID = 1000;

    public static void enqueueWork(Context context, Intent work) {
        enqueueWork(context, DownloadService.class, JOB_ID, work);
    }

    @Override
    protected void onHandleWork(@NonNull Intent intent) {
        // Download the file here
        String fil
        // ...
    }
}
```

```
Intent intent = new Intent(this, DownloadService.class);
intent.putExtra("fileUrl", "https://example.com/file.txt");
DownloadService.enqueueWork(this, intent);
```

Differences

JobIntentService

- Allows it to process multiple intents concurrently
- Is designed to work with Android's power-saving features
- Provides automatic retries if the work fails due to errors or system interruptions

IntentService

- uses a single thread, process each intent sequentially
- Does not provide this functionality
- Does not provide this functionality

Differences

```
public class MyJobIntentService extends JobIntentService {

    @Override
    protected void onHandleWork(@NonNull Intent intent) {
        // This code will be executed on a background thread
        Log.d(TAG, "onHandleWork: Thread id = " + Thread.currentThread().getId());

        // Simulate doing some work that takes time
        try {
            Thread.sleep(5000);
        } for (int i = 1; i <= 5; i++) {
            Intent workIntent = new Intent(this, MyJobIntentService.class);
        }   MyJobIntentService.enqueueWork(this, workIntent);
    }     }
}
```

Differences

```
public class MyIntentService extends IntentService {
    @Override
    protected void onHandleIntent(Intent intent) {
        // This code will be executed on a worker thread
        Log.d(TAG, "onHandleIntent: Thread id = " + Thread.currentThread().getId());

        // Simulate doing some work that takes time
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.pri
        }
    }
}
```

```
for (int i = 1; i <= 5; i++) {
    Intent intent = new Intent(this, MyIntentService.class);
    startService(intent);
}
```

Differences

JobIntentService

onHandleWork: Thread id = 11613

onHandleWork: Thread id = 11614

onHandleWork: Thread id = 11616

onHandleWork: Thread id = 11615

onHandleWork: Thread id = 11617

IntentService

onHandleIntent: Thread id = 14927

onHandleIntent: Thread id = 14927

onHandleIntent: Thread id = 14927

onHandleIntent: Thread id = 14927

onHandleIntent: Thread id = 14927

Foreground Service

Foreground Service

- A foreground service is a type of service that performs a task which must continue running even when the app is in the background or the device is locked
- Requires a persistent notification to be displayed in the notification area, indicating that the service is running.
- This notification cannot be dismissed by the user.

Foreground Service

Examples of tasks that can be performed by foreground services such as:

- Playing audio or video, such as music or podcasts.
- Downloading or uploading files, such as photos or videos.
- Recording audio or video, such as for voice or video calls.

Foreground Service Implementation

```
public class MyForegroundService extends Service {
    private static final int NOTIFICATION_ID = 123;
    private Handler handler;
    private int counter = 0;

    public void onCreate() {}

    public int onStartCommand(Intent intent, int flags, int startId) {}

    public void onDestroy() { }

    public IBinder onBind(Intent intent) { }

    private void runTask() {}

    private Notification getNotification() {}
}
```

```
Intent intent = new Intent(MainActivity.this, MyService.class);
startForegroundService(intent);
```

Foreground Service Implementation

```
@Override
public void onCreate() {
    super.onCreate();
    handler = new Handler();
}

@Override
public void onDestroy() {
    super.onDestroy();
    handler.removeCallbacksAndMessages(null);
    stopForeground(true);
}
```

Foreground Service Implementation

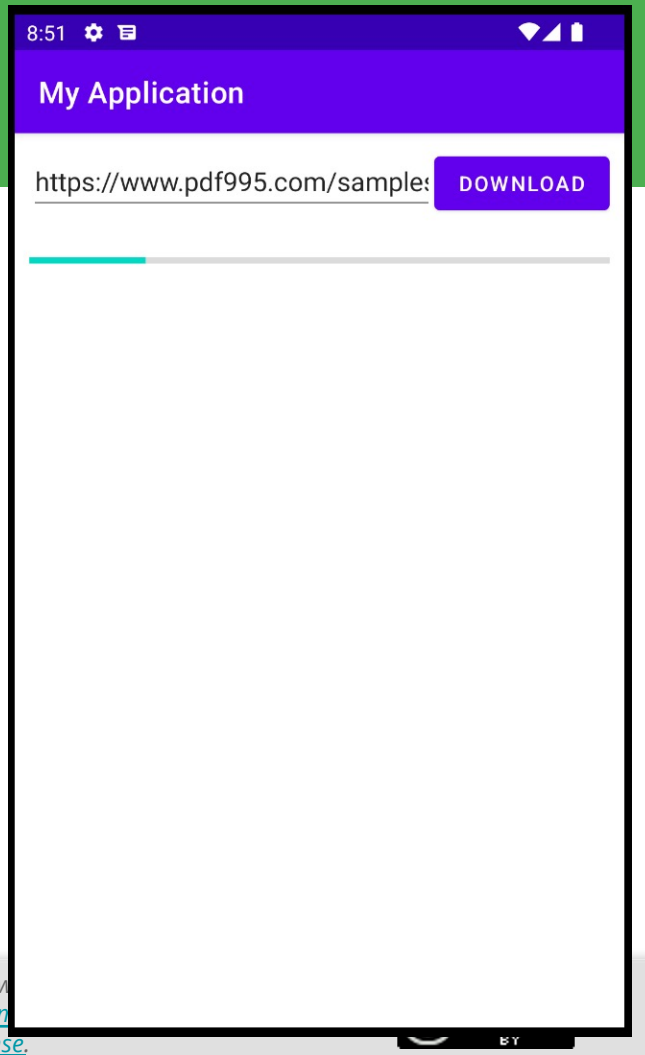
```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    Notification notification = createNotification();
    startForeground(NOTIFICATION_ID, notification);
    runTask();
    return START_STICKY;
}
private void runTask() {
    handler.postDelayed(new Runnable() {
        @Override
        public void run() {
            Log.d("MyForegroundService", "Counter: " + counter);
            counter++;
            handler.postDelayed(this, 1000);
        }
    }, 1000);
}
```

Foreground Service Implementation

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
    NotificationChannel channel = new NotificationChannel("my_channel_id", "My Channel",
                                                          NotificationManager.IMPORTANCE_DEFAULT);
    NotificationManager manager = getSystemService(NotificationManager.class);
    manager.createNotificationChannel(channel);
}

Intent activityIntent = new Intent(this, MainActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, activityIntent,
                                                        PendingIntent.FLAG_IMMUTABLE);
Notification notification = new NotificationCompat.Builder(this, "my_channel_id")
        .setContentTitle("My Foreground Service")
        .setContentText("Running...")
        .setContentIntent(pendingIntent)
        .setSmallIcon(R.drawable.ic_launcher_foreground)
        .build();
```

Demo Foreground Service

```
final int CHUNK_SIZE = 1024;

OkHttpClient client = new OkHttpClient();
Request request = new Request.Builder()
        .url(downloadUrl)
        .build();
Response response = client.newCall(request).execute();
long totalSize = response.body().contentLength();

byte[] buffer = new byte[CHUNK_SIZE];
int bytesRead;
long bytesDownloaded = 0;
```



Sample file: https://www.pdf995.com/samples/pdf.pdf

Demo Foreground Service

```
int nextSize = (int) Math.min(CHUNK_SIZE, totalSize - bytesDownloaded);
long nextOffset = bytesDownloaded + nextSize - 1;
request = new Request.Builder()
        .url(downloadUrl)
        .header("Range", "bytes=" + bytesDownloaded + "-" + nextOffset)
        .build();

response = client.newCall(request).execute();
bytesRead = response.body().byteStream().read(buffer);
bytesDownloaded += bytesRead;

int progress = (int) (bytesDownloaded * 100/totalSize);
Intent intent = new Intent("UPDATE_DOWNLOAD_PROGRESS");
intent.putExtra("progress", progress);
sendBroadcast(intent);
```

```
do … while (bytesRead != -1
            && bytesDownloaded < totalSize)
```

Demo Foreground Service

```
IntentFilter filter = new IntentFilter();
filter.addAction("UPDATE_DOWNLOAD_PROGRESS");
registerReceiver(new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        int progress = intent.getIntExtra("progress", 0);
        progressBar.setProgress(progress);
        Log.d("MainActivity", progress + "");
    }
}, filter);
```

MainActivity.java

Learn more

- Services overview
- Background Execution Limits

What's Next?

- Concept Chapter: 7.4 Services
- No practical

END