第 5 章
Android 服务

5.1 服务概述

服务(Service)是 Android 的 4 大组件之一,用于在后台执行长时间运行的操作。与 Activity 不同,服务程序没有用户界面。启动后,服务程序可以在后台处理不需要用户交互的事务。如果需要,可以在后台运行较长时间,即使在用户切换到其他应用后服务依然可以在后台运行。此外,组件可以绑定到服务以与其交互,甚至执行进程间通信(IPC)。

在 Android 开发中,使用 Activity 可以完成绝大多数的开发内容,但是也会因为某些特定的任务导致不便,例如,需要持续运行的后台任务(如音乐播放、文件下载、定时数据更新)可能因用户切换应用而中断,而服务则能够在后台保持任务的连续性,即使应用界面关闭也不受影响。当任务需要定期或延迟执行时,服务也比 Activity 更稳定,因为它能在屏幕关闭后继续运行,避免被系统回收。

此外,某些与用户界面无关的操作(如文件 I/O、网络请求或批量数据库操作)在 Activity 中执行可能导致界面卡顿,而服务因不依赖界面,能够更高效地完成这些任务。

服务支持进程间通信(IPC),允许不同应用组件共享数据或功能,适合需要绑定的场景。对于需要更高优先级的任务,还可以通过前台服务实现,在通知栏中保持其持续运行,从而降低被系统回收的风险。这些特性使得服务在处理长时间、不需要用户交互的任务时,比 Activity 更加合适。

本章学习目标:

1. 知识理解

- 理解 Android 服务的基本概念及其在 Android 应用中的作用。
- 掌握 Android 服务的分类,包括本地服务、前台服务、后台服务和绑定服务,并理解每种服务的特点和适用场景。
- 理解服务的生命周期,包括服务的启动、运行和停止过程,以及各生命周期方法的作用。
- 理解如何声明和使用服务,掌握在 AndroidManifest.xml 中注册服务的方法。
- 理解系统服务(如 LocationManager、AlarmManager、NotificationManager)的基本概念及其在应用中的使用场景。

- 掌握远程服务的概念，并理解如何使用 AIDL（Android Interface Definition Language）来实现进程间通信。

2. 技能应用能力

- 能够正确地声明服务并在代码中启动、停止和绑定服务。
- 能够实现后台服务，处理长时间运行的任务，如数据同步或定时任务。
- 能够调用系统服务（如 LocationManager、AlarmManager、NotificationManager）并在应用中实现相应的功能。
- 能够实现远程服务，使用 AIDL 进行进程间通信，确保不同应用或进程间的数据交换和操作。

3. 分析与解决问题能力

- 能够分析不同服务类型的优缺点，针对应用场景合理选择服务类型并高效实现。
- 能够根据不同的服务需求，合理使用系统服务（如位置服务、通知管理和定时任务）。
- 能够解决远程服务通信中的常见问题，确保不同进程间的数据传输和操作的安全性和高效性。

5.1.1 服务分类

服务分为本地服务（Local Service）与远程服务（Remote Service）。本地服务是在与应用程序相同的进程中运行的服务，它只能够被同一应用程序的组件（如活动、广播接收器等）绑定和使用。本地服务与应用程序的其他组件共享相同的内存空间和进程环境，因而可以直接访问应用程序中的数据和资源。由于没有进程间通信（IPC）的开销，本地服务的调用速度较快，性能较好。实现本地服务的复杂度较低，通常只需要继承 Service 类并实现相关的生命周期方法。

远程服务是在独立的进程中运行的服务，它可以被其他应用程序的组件通过进程间通信（IPC）机制绑定和使用。远程服务具有自己独立的内存空间，其他应用无法直接访问远程服务的内存数据。由于运行在不同进程中，客户端和服务之间的通信需要通过 AIDL（Android Interface Definition Language）或 Messenger 来实现。远程服务隔离了内存空间，这在某些安全性要求较高的场景中是有利的。远程服务的实现相对复杂，需要处理进程间通信和序列化/反序列化等问题。

因此，本地服务一般适用于同一 App 应用内的服务，运行在应用的主进程中，通信简单且高效。而远程服务适用于需要跨进程或跨 App 应用访问的服务，运行在独立的进程中，使用 AIDL 或 Messenger 实现跨进程通信。

5.1.2 本地服务

本地服务主要有三种类型：前台服务、后台服务和绑定服务。每种类型的服务有其特定的用途和行为。

1. 前台服务

前台服务是一种对用户可见的服务，会在通知栏显示一个持续通知，通常用于执行用户持续关注感知的任务，如播放音乐、下载文件、位置跟踪等。前台服务在系统资源紧张时具有较高的优先级，不容易被系统回收。前台服务必须显示 Notification。即使用户没有与应用

互动，前台服务也会继续运行。服务必须调用 startForeground()，且在 AndroidManifests. xml 中声明 foregroundServiceType，优先级较高，不易被系统杀死，适合需要长期运行的任务。

2. 后台服务

后台服务是一种在没有用户界面的情况下运行的服务，执行用户不会直接注意到的操作。系统资源紧张时，后台服务有可能被终止。适用于不需要用户关注的后台任务，如数据同步或文件下载。可以使用 startService()调用后台服务直接在后台运行。后台服务优先级较低，系统资源紧张时可能会被终止，适合短期任务。

3. 绑定服务

绑定服务是一种允许应用组件(如 Activity)绑定并与之进行交互的服务。客户端通过 bindService()方法绑定服务，并通过 onServiceConnected()回调方法获取 IBinder 对象，从而与服务进行通信。适用于需要与服务进行密切交互的任务，如提供计算服务或访问远程资源。

5.2　服务的生命周期

生命周期(Lifecycle)指的是一个对象或组件从创建到销毁所经历的不同阶段或状态变化过程。在 Android 开发中，生命周期特别重要，因为它帮助开发者理解和管理组件的运行状态，确保资源在正确的时间被初始化和释放，避免内存泄漏、性能损耗或系统崩溃等问题。

在 Android 服务中，服务生命周期指的是服务在其存在期间经历的不同状态转换，包括创建、启动、运行、绑定和销毁等过程。服务的生命周期决定了服务何时启动、何时终止、如何响应绑定请求，以及如何正确管理资源。掌握这些状态和相应的生命周期方法可以帮助开发者在适当的位置与方法中初始化程序和释放资源，避免内存泄漏和不必要的资源占用，确保服务在后台的稳定性和高效性。服务的生命周期方法包括 onCreate()、onStartCommand()、onBind()和 onDestroy()等，每个方法在服务的不同状态下被调用。创建 Android 服务时需要通过继承 Service 类来实现，在服务类中重写生命周期的方法实现相应功能，相关代码如下。

```
public class MyService extends Service {
    public MyService () {
    }
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    @Override
    public void onCreate() {
        super.onCreate();
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        return super.onStartCommand(intent, flags, startId);
    }
    @Override
    public void onDestroy() {
```

```
        super.onDestroy();
    }
    @Override
    public boolean onUnbind(Intent intent) {
        return super.onUnbind(intent);
    }
}
```

Service 类的生命周期从 onCreate()开始，到 onDestroy()结束。生命周期中的常用方法如下。

(1) IBinder onBind(Intent intent)：当组件(如 Activity)通过 bindService()绑定服务时，系统会调用此方法。它用于返回一个 IBinder 对象，供客户端与服务进行通信。如果服务不支持绑定，返回 null。

(2) void onCreate()：服务首次创建时调用。这个方法只会在服务的整个生命周期中被调用一次，用于执行一次性的初始化操作，如创建必要的资源。

(3) int onStartCommand(Intent intent,int flags,int startId)：每次通过 startService()启动服务时，都会调用这个方法。用于处理启动服务时的任务。返回值决定了如果服务被系统终止后如何重启。

(4) void onDestroy()：服务被停止或销毁时调用。用于清理资源，如关闭线程、停止播放、取消通知等。

(5) boolean onUnbind(Intent intent)：当所有客户端都解除绑定时调用。用于处理服务的解绑操作。返回值 true 表示如果有新的客户端再次绑定，onRebind()会被调用；false 表示不会再调用 onRebind()。

(6) void onRebind(Intent intent)：当新的客户端绑定到已经解除绑定但仍在运行的服务时调用。这个方法只有在之前的 onUnbind()返回 true 时才会被调用。

(7) void onLowMemory()：当系统内存不足，需要释放资源时调用。此方法提示应用减少内存使用，例如，清理缓存或关闭不必要的服务。

(8) void onTrimMemory(int level)：当系统内存使用达到某个级别时调用，level 参数表示系统的内存状态。根据 level 的不同，应用程序应该相应地减少内存使用。例如，释放无用的对象、减少缓存大小等。

(9) void onTaskRemoved(Intent rootIntent)：当任务(通常是一个 Activity)被用户移除时调用。如果服务与该任务相关联，可以通过这个方法知道任务已被删除，并执行相关的清理工作。

(10) void onTimeout(int startId)：用于处理前台服务在特定情况下的超时回调。该方法主要用于防止前台服务长时间占用系统资源，而未能完成其任务。

服务的生命周期流程如图 5.1 所示。

使用前台服务时，服务的生命周期为 onCreate→onStartCommand→onDestroy，Activity 通过 startForegroundService(Intent intent)启动前台服务，此时若为第一次启动，调用服务生命周期中的 onCreate()方法，之后，无论是否为第一次启动，均调用 onStartCommand(Intent intent,int flags,int startId)方法。然后在服务类中调用 startForeground(int id,Notification

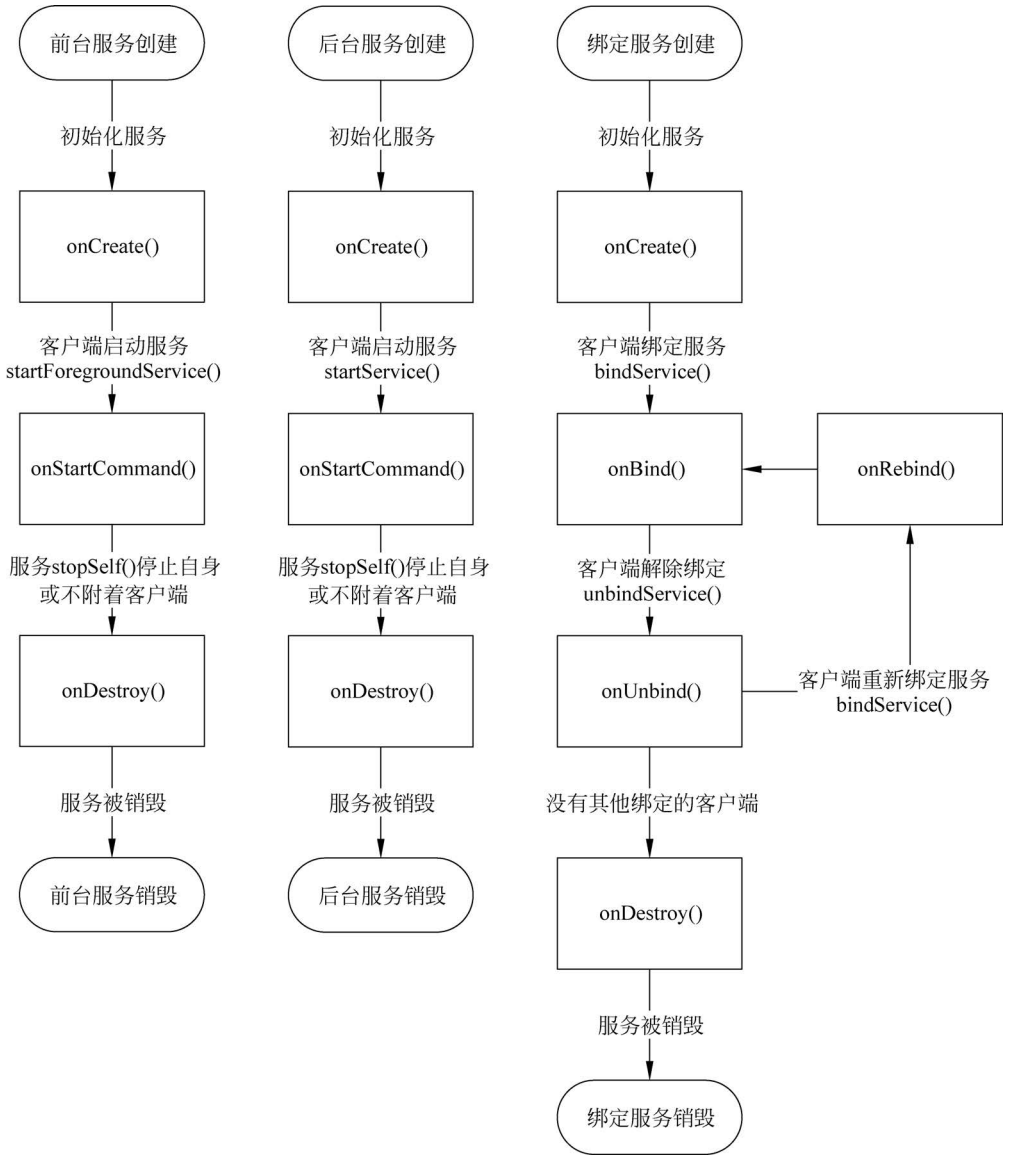

图 5.1　服务的生命周期流程

notification)将服务提升为前台服务,此方法必须在启动后短时间内调用,否则系统将判定服务未响应并停止服务。当前台服务运行结束时,Activity 通过调用 stopForeground(boolean removeNotification)停止前台状态。要彻底停止服务,需要在服务类通过 stopSelf()停止服务,此时调用 onDestroy()方法,服务被销毁,生命周期结束。

后台服务与前台服务类似,生命周期为 onCreate→onStartCommand→onDestroy,Activity 通过 startService(Intent intent)启动后台服务,此时若为第一次启动,调用生命周期中的 onCreate()方法。之后,无论是否为第一次启动,均调用 onStartCommand(Intent intent,int flags,int startId)方法。当后台服务需要停止时,服务类通过 stopSelf()或 Activity 调用 stopService(Intent intent)停止服务,此时服务的生命周期结束。

绑定服务与上面的服务有所不同,生命周期为 onCreate→onBind→onUnbind→onDestroy,

Activity 通过 bindService(Intent intent,ServiceConnection conn,int flags)绑定服务,此时若服务第一次启动,调用 onCreate()方法。之后,无论是否为第一次启动,调用 onBind(Intent intent)方法返回一个 IBinder 对象,客户端通过 ServiceConnection 接收。服务在绑定期间运行,多个客户端可以绑定同一个服务。当需要解除绑定时,Activity 通过 unbindService(ServiceConnection conn)解绑服务,服务类 onUnbind(Intent intent)方法被调用,之后若重新进行绑定,Activity 依旧使用 bindService(Intent intent,ServiceConnection conn,int flags)绑定服务,但服务类中会在 onBind(Intent intent)前先调用 onRebind()方法来重新绑定服务,再调用 onBind(Intent intent)方法。当没有客户端绑定时,服务可能会被销毁,调用 onDestroy()方法。

5.3 服务的使用

5.3.1 服务声明

在使用服务前,需要在 AndroidManifests.xml 内对服务进行声明才能正常使用,在 AndroidManifests.xml 中,service 应当在＜application＞标签中定义服务类所对应的＜service＞标签,并设置相关属性才能正常使用,服务声明的语法如下。

```
<service android:description="string resource"
      android:directBootAware=["true" | "false"]
      android:enabled=["true" | "false"]
      android:exported=["true" | "false"]
      android:foregroundServiceType=["camera" | "connectedDevice" |
                                     "dataSync" | "health" | "location" |
                                     "mediaPlayback" | "mediaProjection" |
                                     "microphone" | "phoneCall" |
                                     "remoteMessaging" | "shortService" |
                                     "specialUse" | "systemExempted"]
      android:icon="drawable resource"
      android:isolatedProcess=["true" | "false"]
      android:label="string resource"
      android:name="string"
      android:permission="string"
      android:process="string" >
    ...
</service>
```

AndroidManifests.xml 声明的服务属性含义如表 5.1 所示。

表 5.1 AndroidManifests.xml 中服务的属性

属　性	功 能 说 明
android:description	服务的描述
android:directBootAware	确定服务是否可感知直接启动,是否可以在用户解锁设备之前运行。默认值为否
android:enabled	确定系统是否可以实例化服务。默认值为是

续表

属　　性	功 能 说 明
android:exported	确定其他应用的组件是否可以调用服务或与之交互。默认值为否
android:foregroundServiceType	阐明服务是满足特定用例要求的前台服务
android:icon	服务的图标
android:isolatedProcess	是否在与系统其余部分隔离的特殊进程下运行
android:label	服务的用户可读名称
android:name	Service 子类，如 com. servicedemo. Service
android:permission	实体启动服务或绑定到服务所需的权限的名称
android:process	运行服务的进程的名称

5.3.2　前台服务

前台服务是对用户可见的服务，通常用于执行用户持续关注的重要任务。启动前台服务时，系统会要求服务显示一个持续的通知，用户可以通过通知栏看到服务的运行状态。例如，音乐播放、导航、计步等功能常常会使用前台服务。

启动前台服务的步骤如下。

(1) 创建一个服务类，继承 Service，并实现相关生命周期方法。

(2) 构建一个通知对象。前台服务要求通过通知显示状态信息，因此必须构建一个 Notification 对象。

(3) 调用 startForeground()方法。在 onStartCommand()方法中使用 startForeground()，传入通知的 ID 和通知对象，将服务提升到前台状态。

(4) 停止前台服务。当任务完成或服务不再需要运行时，可以调用 stopForeground (true)以取消前台状态，并调用 stopSelf()或 stopService()结束服务。

【例 5.1】 前台服务的示例。本例中使用前台服务进行播放音乐，并提供播放与暂停的功能。播放的音乐 sample_music. mp3 放置于 res 目录下 raw 资源文件夹中。

示例中 App 的配置文件 AndroidManifests. xml 代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
    <uses-permission android:name="android.permission.POST_NOTIFICATIONS" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Chapter5">
        <service
            android:name=".ForegroundServiceDemo"
            android:foregroundServiceType="shortService"
            android:enabled="true" />
        <activity
```

```
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

布局文件 ForegroundServiceDemo\activity_main. xml 中的代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <Button
        android:id="@+id/btn_start_foreground_service"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="启动/停止前台服务" />
</LinearLayout>
```

Activity 程序 ForegroundServiceDemo\MainActivity. java 代码如下。

```
package com.example.foregroundservicedemo;
public class MainActivity extends AppCompatActivity {
    private boolean Foreground = false;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
        //设置窗口内边距以适应系统栏
         ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main),
(v, insets) -> {
            Insets systemBars = insets
                .getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top,
                systemBars.right, systemBars.bottom);
            return insets;
        });
        Button btnStartForegroundService =
            findViewById(R.id.btn_start_foreground_service);
        //创建通知渠道(仅在 Android O 以上版本需要)
```

```
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU)
        {
                //请求通知权限(仅在 Android T 以上版本需要)
                if (checkSelfPermission(Manifest.permission.POST_NOTIFICATIONS) !=
PackageManager.PERMISSION_GRANTED) {
                    requestPermissions(new String[]{Manifest.permission.POST_
NOTIFICATIONS}, 101);
                }
            }
            String channelId = "service_demo";
            CharSequence name = "服务示例";
            String description = "服务通知示例";
                 int NotificationChannel channel = new NotificationChannel
(channelId, name, channel.setDescription(description);
                NotificationManager notificationManager = getSystemService
(NotificationManager.class);
            notificationManager.createNotificationChannel(channel);
        }
        //设置按钮点击事件以启动或停止前台服务
        btnStartForegroundService.setOnClickListener(v -> {
            if (Foreground) {
                //停止前台服务
                Intent intent = new Intent(this, ForegroundServiceDemo.class);
                stopService(intent);
                Foreground = false;
            } else {
                //启动前台服务
                Intent intent = new Intent(this, ForegroundServiceDemo.class);
                if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
                    startForegroundService(intent);
                } else {
                    startService(intent);
                }
                Foreground = true;
            }
        });
    }
}
```

服务类程序 ForegroundServiceDemo\ForegroundServiceDemo. java 代码如下。

```
package com.example.foregroundservicedemo;
public class ForegroundServiceDemo extends Service {
    private MediaPlayer mediaPlayer;
    private static final String ACTION_PLAY =
        "com.example.foregroundservicedemo.ACTION_PLAY";
    private static final String ACTION_STOP =
        "com.example.foregroundservicedemo.ACTION_STOP";
```

```
    public ForegroundServiceDemo() {
    }
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    @Override
    public void onCreate() {
        super.onCreate();

        //创建播放操作意图
        Intent playIntent = new Intent(this, ForegroundServiceDemo.class);
        playIntent.setAction(ACTION_PLAY);
        PendingIntent playPendingIntent = PendingIntent.getService(this, 0,
playIntent, PendingIntent.FLAG_IMMUTABLE);
        //创建停止操作意图
        Intent stopIntent = new Intent(this, ForegroundServiceDemo.class);
        stopIntent.setAction(ACTION_STOP);
        PendingIntent stopPendingIntent = PendingIntent.getService(this, 0,
stopIntent, PendingIntent.FLAG_IMMUTABLE);
        //创建通知
        Notification notification = new NotificationCompat.Builder(this,
"service_demo")
                .setContentTitle("前台服务示例")
                .setContentText("服务正在运行中")
                .setSmallIcon(R.drawable.music)
                .setPriority(NotificationCompat.PRIORITY_DEFAULT)
                .addAction(R.drawable.play, "播放", playPendingIntent)
                .addAction(R.drawable.pause, "停止", stopPendingIntent)
                .build();
        //将服务置于前台并显示通知
        startForeground(1, notification);
        //初始化媒体播放器
        mediaPlayer = MediaPlayer.create(this, R.raw.sample_music);
        mediaPlayer.setLooping(true);
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        if (intent != null) {
            String action = intent.getAction();
            if (ACTION_PLAY.equals(action)) {
                //播放音乐
                if (!mediaPlayer.isPlaying()) {
                    mediaPlayer.start();
                    Log.i("前台服务示例", "音乐播放");
                }
            } else if (ACTION_STOP.equals(action)) {
                //暂停音乐
                if (mediaPlayer.isPlaying()) {
```

```
                    mediaPlayer.pause();
                    Log.i("前台服务示例", "音乐暂停");
                }
            }
        }
        return super.onStartCommand(intent, flags, startId);
    }
    @Override
    public void onDestroy() {
        Log.i("前台服务示例", "前台服务停止");
        if (mediaPlayer != null) {
            //停止并释放媒体播放器
            mediaPlayer.stop();
            mediaPlayer.release();
            mediaPlayer = null;
        }
        super.onDestroy();
    }
}
```

App 程序在执行时需要用户授权相关权限，如图 5.2 所示。用户单击启动前台服务按钮，程序调用服务类执行音乐播放，如图 5.3 所示。

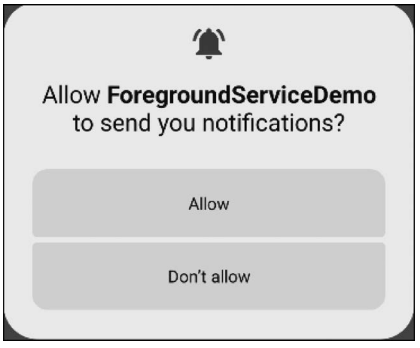

图 5.2　申请通知权限

图 5.3　前台服务通知

服务类调用的信息在 Logcat 窗口中显示，如图 5.4 所示。

图 5.4　前台服务 Log 输出

5.3.3 后台服务

在 Android 中，启动后台服务和前台服务的方式有所不同，且两者适用于不同的应用场景。后台服务通常用于在没有用户界面的情况下执行短期或无须用户关注的任务。这些任务通常在后台执行，并不显示在通知栏中。后台服务适合执行如数据同步、清理缓存、定期更新等无须用户干预的操作。

启动后台服务的步骤如下。

(1) 创建服务类，继承 Service 类，实现相关生命周期方法。

(2) 调用 startService()方法启动服务。通过调用 startService(Intent intent)启动服务，这会触发服务的 onStartCommand()方法。

(3) 停止后台服务。任务完成后，后台服务应当主动调用 stopSelf()方法结束，或由外部调用 stopService()停止服务。

【例 5.2】 后台服务使用示例。通过调用后台服务来生成一个 0～100 的随机数并通过广播传回给主界面，输出到相关控件中。

在 AndroidManifests.xml 文件中配置服务类 BackgroundServiceDemo 的注册。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Chapter5">
        <service
            android:name=".BackgroundServiceDemo"
            android:enabled="true"
            android:exported="true" >
            <intent-filter>
                <action android:name="com.example.backgroundservicedemo.ACTION_
START_SERVICE" />
            </intent-filter>
        </service>
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

App 界面布局 BackgroundServiceDemo\activity_main.xml 代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <Button
        android:id="@+id/btn_start_background_service"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="启动后台服务" />
    <Button
        android:id="@+id/btn_start_implicit_background_service"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="启动后台服务(隐式)" />
    <TextView
        android:id="@+id/text_view_random_number"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="随机数：" />
</LinearLayout>
```

MainActivity 程序 BackgroundServiceDemo\MainActivity.java 代码如下。

```
package com.example.backgroundservicedemo;
public class MainActivity extends AppCompatActivity {
    private TextView textViewRandomNumber;
    private BroadcastReceiver randomNumberReceiver;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main),
(v, insets) -> {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.
systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom);
            return insets;
        });
        textViewRandomNumber = findViewById(R.id.text_view_random_number);
        //设置按钮以启动后台服务
          Button btnStartBackgroundService = findViewById(R.id.btn_start_
background_service);
        btnStartBackgroundService.setOnClickListener(v -> {
```

```
            Intent intent = new Intent(this, BackgroundServiceDemo.class);
            startService(intent);
        });
        //设置按钮以使用隐式意图启动后台服务
        Button btnStartImplicitBackgroundService = findViewById(R.id.btn_start_
implicit_background_service);
        btnStartImplicitBackgroundService.setOnClickListener(v -> {
            Intent intent = new Intent("com.example.backgroundservicedemo.
ACTION_START_SERVICE");
            intent.setPackage(getPackageName());
            startService(intent);
        });
        //注册 BroadcastReceiver 以接收随机数广播
        randomNumberReceiver = new BroadcastReceiver() {
            @Override
            public void onReceive(Context context, Intent intent) {
                if (BackgroundServiceDemo.ACTION_RANDOM_NUMBER.equals(intent.
getAction())) {
                    int randomNumber =intent.getIntExtra(BackgroundServiceDemo.
EXTRA_RANDOM_NUMBER, -1);
                    textViewRandomNumber.setText("随机数：" + randomNumber);
                }
            }
        };
         IntentFilter filter = new IntentFilter(BackgroundServiceDemo.ACTION_
RANDOM_NUMBER);
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
             registerReceiver(randomNumberReceiver, filter, Context.RECEIVER_
EXPORTED);
        }
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        //注销 BroadcastReceiver 以避免内存泄漏
        unregisterReceiver(randomNumberReceiver);
    }
}
```

服务类程序 BackgroundServiceDemo\ BackgroundServiceDemo.java 代码如下。

```
package com.example.backgroundservicedemo;
public class BackgroundServiceDemo extends Service {
    public static final String ACTION_RANDOM_NUMBER = "com.example.
backgroundservicedemo.ACTION_RANDOM_NUMBER";
    public static final String EXTRA_RANDOM_NUMBER = "com.example.
backgroundservicedemo.EXTRA_RANDOM_NUMBER";
    public BackgroundServiceDemo() {
    }
```

```
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    @Override
    public void onCreate() {
        super.onCreate();
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        Log.i("后台服务示例", "后台服务启动");
        //生成随机数并发送广播
        Random random = new Random();
        int randomNumber = random.nextInt(100);
        Intent broadcastIntent = new Intent(ACTION_RANDOM_NUMBER);
        broadcastIntent.putExtra(EXTRA_RANDOM_NUMBER, randomNumber);
        sendBroadcast(broadcastIntent);
        Log.i("后台服务示例", "随机数：" + randomNumber);
        return super.onStartCommand(intent, flags, startId);
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
    }
}
```

程序运行结果如图 5.5 所示。

图 5.5　后台服务运行示例

程序运行时，打开 Logcat 窗口可以查看后台服务生命周期方法执行的输出信息，如图 5.6 所示。

图 5.6　Logcat 窗口查看输出信息

5.3.4 绑定服务

绑定服务提供了客户端与服务之间的通信渠道，允许客户端调用服务中的方法，获取服务状态或执行操作。多个客户端可同时绑定到该服务。bindService()方法用于绑定一个服务，使应用的组件(如 Activity)能够与服务进行交互。当客户端完成与服务的交互后，会调用 unbindService()来解除绑定。如果没有绑定到服务的客户端，则系统会销毁该服务。

【例 5.3】 绑定服务示例。在 Activity 中显示绑定服务按钮，服务绑定成功后，调用服务计算两个输入框的输入数值的和，并将结果输出到主界面中。

在 AndroidManifests.xml 文件中注册服务类 BindServiceDemo。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Chapter5">
        <service
            android:name="com.example.bindservicedemo.BindServiceDemo"
            android:enabled="true" />
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

界面布局 BindServiceDemo\activity_main.xml 代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <!-- 输入参数 1 -->
    <EditText
        android:id="@+id/et_param1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
        android:hint="参数 1"
        android:inputType="number" />
    <!-- 输入参数 2 -->
    <EditText
        android:id="@+id/et_param2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="参数 2"
        android:inputType="number" />
    <!-- 绑定/解绑服务按钮 -->
    <Button
        android:id="@+id/btn_bind_service"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="绑定/解绑服务"/>
    <!-- 计算参数和按钮 -->
    <Button
        android:id="@+id/btn_calculate"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="计算参数和"/>
    <!-- 显示结果 -->
    <TextView
        android:id="@+id/tv_result"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="请绑定服务" />
</LinearLayout>
```

MainActivity 程序 BindServiceDemo\MainActivity.java 代码如下。

```
package com.example.bindservicedemo;
public class MainActivity extends AppCompatActivity {
    private boolean isBound = false;
    private BindServiceDemo bindService;
    //定义 ServiceConnection 对象
    private final ServiceConnection connection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            BindServiceDemo.LocalBinder binder = (BindServiceDemo.LocalBinder)
service;
            bindService = binder.getService();
            isBound = true;
        }
        @Override
        public void onServiceDisconnected(ComponentName name) {
            isBound = false;
        }
    };
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
         ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main),
(v, insets) -> {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.
systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom);
            return insets;
        });
        //获取 UI 控件
        EditText etParam1 = findViewById(R.id.et_param1);
        EditText etParam2 = findViewById(R.id.et_param2);
        TextView tvResult = findViewById(R.id.tv_result);
        Button btnBindService = findViewById(R.id.btn_bind_service);
        Button btnCalculate = findViewById(R.id.btn_calculate);
        //绑定/解绑服务按钮点击事件
        btnBindService.setOnClickListener(v -> {
            if (isBound) {
                unbindService(connection);
                isBound = false;
                tvResult.setText("服务已解绑,请重新绑定服务");
            } else {
                Intent intent = new Intent(this, BindServiceDemo.class);
                bindService(intent, connection, BIND_AUTO_CREATE);
                if (isBound) {
                    tvResult.setText("服务已绑定");
                } else {
                    tvResult.setText("服务绑定失败");
                }
            }
        });
        //计算按钮点击事件
        btnCalculate.setOnClickListener(v -> {
            if (isBound) {
                int param1 = Integer.parseInt(etParam1.getText().toString());
                int param2 = Integer.parseInt(etParam2.getText().toString());
                int result = bindService.calculateSum(param1, param2);
                tvResult.setText("结果: " + result);
            } else {
                tvResult.setText("服务未绑定");
            }
        });
    }
}
```

服务类程序 bindservicedemo\BindServiceDemo.java 代码如下。

```
package com.example.bindservicedemo;
public class BindServiceDemo extends Service {
    public BindServiceDemo() {
    }
    //创建 Binder 对象
    private final IBinder binder = new LocalBinder();
    //定义 LocalBinder 类
    public class LocalBinder extends Binder {
        BindServiceDemo getService() {
            return BindServiceDemo.this;
        }
    }
    @Override
    public IBinder onBind(Intent intent) {
        Log.i("绑定服务示例", "服务已绑定");
        return binder;
    }
    @Override
    public boolean onUnbind(Intent intent) {
        Log.i("绑定服务示例", "服务已解绑");
        return super.onUnbind(intent);
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
    }
    //计算两个参数的和
    public int calculateSum(int param1, int param2) {
        return param1 + param2;
    }
}
```

程序的运行结果如图 5.7 所示。运行程序后需要先单击“绑定/解绑服务”按钮，绑定服务成功后，输入计算数据，单击“计算参数和”按钮，调用服务类进行计算，并将计算结果显示在界面上。

图 5.7　绑定服务运行示例

程序运行时，打开 Logcat 窗口可以查看后台服务生命周期方法执行的输出信息，如图 5.8 所示。

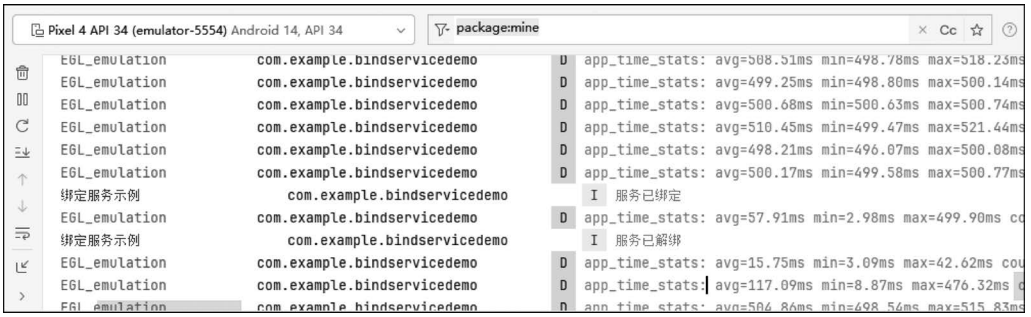

图 5.8 绑定服务 Log 输出

5.4 调用系统服务

Android 系统提供了多种系统服务，帮助开发者访问设备的核心功能和硬件资源。这些服务通过 Context.getSystemService()方法获取，并提供特定的接口用于执行相关操作。几种常见的系统服务及其含义如表 5.2 所示。

表 5.2 主要的系统服务及其含义

系统服务名	功 能
ActivityManager	获取与运行的应用程序相关的信息，如任务、服务和进程信息
AlarmManager	计划在未来某个时间点执行的操作(如闹钟、定时任务)
AudioManager	控制和管理音频流和音量
ClipboardManager	访问和操作剪贴板中的内容
ConnectivityManager	管理网络连接，检查网络状态和网络连接类型
LocationManager	访问地理位置服务，获取 GPS 或网络位置
NotificationManager	管理通知，发布和取消通知
PowerManager	控制设备电源管理，如唤醒锁(WakeLock)
SensorManager	访问设备的传感器，如加速度计、陀螺仪、光传感器等
TelephonyManager	管理电话功能，如获取 SIM 卡信息、电话状态等
Vibrator	用于控制设备的振动功能
WifiManager	用于管理 Wi-Fi 连接和配置 Wi-Fi 网络
WindowManager	用于管理应用窗口和显示内容
InputMethodManager	用于管理输入法，控制软键盘的显示和隐藏
PackageManager	用于获取已安装应用的信息、启动应用、获取权限等
BatteryManager	用于访问电池状态和管理电池电量信息
BluetoothManager	用于管理蓝牙连接和设备

5.4.1 LocationManager

LocationManager 用于获取地理位置信息，如 GPS 位置和网络位置。它允许应用程序获取设备的当前位置，以及监听位置变化和地理围栏等功能。

LocationManager 可以使用多种位置提供器来获取设备位置，每种提供器的精度、速度、功耗和可用性都不同。获取设备位置的方式如下。

(1) GPS_PROVIDER：使用 GPS 卫星获取位置信息，精度高，但耗电量大，且在室内可能无法获取到位置。

(2) NETWORK_PROVIDER：使用移动网络或 Wi-Fi 获取位置信息，精度较低，但耗电量低，适用于室内或城市环境。

(3) PASSIVE_PROVIDER：不主动请求位置更新，而是通过其他应用或服务获取到的位置。

LocationManager 类的常用方法如下。

(1) boolean addNmeaListener(OnNmeaMessageListener listener, Handler handler)：添加一个 NMEA 监听器，用于接收 NMEA 消息。监听器将在指定的 Handler 线程上运行。

(2) void addProximityAlert(double latitude, double longitude, float radius, long expiration, PendingIntent pendingIntent)：为指定的经纬度设置一个接近警报。当设备进入或离开指定半径区域时，将触发给定的 PendingIntent。

(3) void getCurrentLocation(String provider, LocationRequest locationRequest, CancellationSignal cancellationSignal, Executor executor, Consumer<Location> consumer)：根据指定的 LocationRequest，从指定的提供者异步获取一次当前位置。

(4) List<GnssAntennaInfo> getGnssAntennaInfos()：返回当前的 GNSS 天线信息列表，如果未知或不支持则返回 null。

(5) GnssCapabilities getGnssCapabilities()：返回 GNSS 芯片支持的功能和特性。

(6) String getGnssHardwareModelName()：返回 GNSS 硬件驱动程序的型号名称，包括供应商和硬件/软件版本。如果不可用，则返回 null。

(7) Location getLastKnownLocation(String provider)：获取指定提供者的最后已知位置，如果没有可用的位置，则返回 null。

(8) boolean isLocationEnabled()：返回定位功能的当前启用或禁用状态。

(9) boolean registerAntennaInfoListener(Executor executor, GnssAntennaInfo. Listener listener)：注册一个 GNSS 天线信息监听器，以接收天线信息的变化。

(10) boolean registerGnssMeasurementsCallback(Executor executor, GnssMeasure-mentsEvent. Callback callback)：注册一个 GNSS 测量回调。

(11) void requestLocationUpdates(String provider, long minTimeMs, float minDistanceM, LocationListener listener)：使用指定的提供者注册位置更新，设置最小时间间隔和最小距离，并在调用线程的 Looper 上回调。

在使用 GPS 前，需要先检查 GPS 服务是否启用，才能正确获得当前的位置，以下是启用 GPS 的步骤。

(1) 检查位置服务是否启用。

```
boolean isGPSEnabled =
    locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);
boolean isNetworkEnabled =
    locationManager.isProviderEnabled(LocationManager.NETWORK_PROVIDER);
```

(2) 获取 LocationManager 实例。

```
LocationManager locationManager = (LocationManager)
    getSystemService(Context.LOCATION_SERVICE);
```

(3) 获取当前位置。

```
Location location =
    locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
if (location != null) {
    double latitude = location.getLatitude();
    double longitude = location.getLongitude();
    //使用位置数据
}
```

(4) 监听位置变化。

```
locationManager.requestLocationUpdates(
    LocationManager.GPS_PROVIDER,
    5000,       //最小时间间隔(毫秒)
    10,         //最小距离间隔(米)
    new LocationListener() {
      @Override
      public void onLocationChanged(Location location) {
          double latitude = location.getLatitude();
          double longitude = location.getLongitude();
          //使用更新的位置信息
      }
      @Override
      public void onStatusChanged(String provider, int status, Bundle extras) {
          //处理提供者状态变化
      }
      @Override
      public void onProviderEnabled(String provider) {
          //处理提供者启用
      }
      @Override
      public void onProviderDisabled(String provider) {
          //处理提供者禁用
      }
    }
);
```

【例 5.4】 使用 LocationManager 类接收位置信息的示例,获取当前位置信息后将信息显示在界面上。

调用定位信息,需要在 AndroidManifests.xml 中添加如下权限。

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

MainActivity 程序 SystemServiceDemo\MainActivity.java 代码如下。

```
package com.example.systemservicedemo;
public class MainActivity extends AppCompatActivity {
    private TextView tvResult;
    private LocationManager locationManager;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tvResult = findViewById(R.id.tv_result);
        Button btnGetLocation = findViewById(R.id.btn_get_location);
        locationManager = (LocationManager) getSystemService(Context.LOCATION_
SERVICE);
        btnGetLocation.setOnClickListener(v -> getLocationAndDisplay());
        //Check and request location permission
          if (ActivityCompat.checkSelfPermission(this, Manifest.permission.
ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
             ActivityCompat.requestPermissions(this, new String[]{Manifest.
permission.ACCESS_FINE_LOCATION}, 1);
        }
    }
    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[]
permissions, @NonNull int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        if (requestCode == 1) {
            if (grantResults.length > 0 && grantResults[0] == PackageManager.
PERMISSION_GRANTED) {
                getLocationAndDisplay();
            } else {
                tvResult.setText("未授予位置权限");
            }
        }
    }
    private void getLocationAndDisplay() {
         if (ActivityCompat.checkSelfPermission(this, Manifest.permission.
ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED) {
              if (locationManager.isProviderEnabled(LocationManager.GPS_
PROVIDER)) {
               Location lastKnownLocation = locationManager.getLastKnownLocation
(LocationManager.GPS_PROVIDER);
              if (lastKnownLocation != null) {
                   String locationText = "当前位置信息: " + lastKnownLocation.
getLatitude() + ", " + lastKnownLocation.getLongitude();
                  tvResult.setText(locationText);
                  Log.i("SystemServiceExample", locationText);
              } else {
                  tvResult.setText("没有最后已知位置");
```

```
                    Log.i("SystemServiceExample", "没有最后已知位置");
                }
            } else {
                tvResult.setText("GPS 未启用");
                Log.i("SystemServiceExample", "GPS 未启用");
            }
        } else {
            tvResult.setText("未授予位置权限");
        }
    }
}
```

App 在执行时需要开启定位权限，如图 5.9 所示。由于是 LocationManager 获取位置信息，因此在获取位置大幅依赖于 GPS 信号，导致在室内难以进行定位，需要携带手机在空旷地带走动一段时间，LocationManager 才能获取位置信息，从而输出上一个已知的位置。

图 5.9　授权定位

运行结果如图 5.10 所示。

图 5.10　定位服务示例

5.4.2　AlarmManager

AlarmManager 用于调用系统的定时任务，在特定的时间点启动指定的操作（如执行某个任务、启动一个服务等）。AlarmManager 可以用于设置定时任务、周期性任务，甚至是在设备休眠时也能唤醒设备执行任务。

AlarmManager 中常用的方法如表 5.3 所示。

表 5.3　AlarmManager 中的常用方法

方　法	功 能 说 明
void setExact(int type,long triggerAtMillis,String tag,AlarmManager.OnAlarmListener listener,Handler targetHandlern)	在指定的时间触发一次性闹钟
void setRepeating(int type,long triggerAtMillis,long intervalMillis,PendingIntent operation)	在指定的时间开始，并以固定的时间间隔重复触发
void setInexactRepeating(int type,long triggerAtMillis,long intervalMillis,PendingIntent operation)	与 etRepeating()类似，但在时间点上不精确，以便节省电量
void setAndAllowWhileIdle(int type,long triggerAtMillis,PendingIntent operation)	在设备处于低功耗模式时触发闹钟
void setExactAndAllowWhileIdle(int type,long triggerAtMillis,PendingIntent operation)	在设备处于低功耗模式时以精确时间触发一次性闹钟

其中，第一个参数 int type 可以定义的常量如表 5.4 所示。

表 5.4　触发类型

类　型	功 能 说 明
RTC	基于 UTC 时间的闹钟，不会唤醒设备
RTC_WAKEUP	基于 UTC 时间的闹钟，唤醒设备触发闹钟
ELAPSED_REALTIME	基于设备启动后经过的时间，不会唤醒设备
ELAPSED_REALTIME_WAKEUP	设备启动后经过的时间，唤醒设备触发闹钟

通过 AlarmManager 设置与管理闹钟的步骤如下。

（1）获取 AlarmManager 实例。

```
AlarmManager alarmManager = (AlarmManager) getSystemService(Context.ALARM_
SERVICE);
```

（2）设置一次性闹钟。

```
Intent intent = new Intent(this, MyBroadcastReceiver.class);
PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 0, intent, 0);
AlarmManager alarmManager = (AlarmManager) getSystemService(Context.ALARM_
SERVICE);
long triggerTime = System.currentTimeMillis() + 60000; //1 分钟后触发
alarmManager.setExact(AlarmManager.RTC_WAKEUP, triggerTime, pendingIntent);
```

(3) 设置重复闹钟。

```
Intent intent = new Intent(this, MyBroadcastReceiver.class);
PendingIntent pendingIntent = PendingIntent.getBroadcast(this, 0, intent, 0);
AlarmManager alarmManager = (AlarmManager) getSystemService(Context.ALARM_
SERVICE);
long interval = AlarmManager.INTERVAL_FIFTEEN_MINUTES;
long triggerTime = System.currentTimeMillis() + interval;
alarmManager.setInexactRepeating(AlarmManager.RTC_WAKEUP, triggerTime,
interval, pendingIntent);
```

(4) 取消闹钟。

```
alarmManager.cancel(pendingIntent);
```

5.4.3 NotificationManager

NotificationManager 用于管理通知,允许应用在通知栏显示通知信息。通知通常显示在状态栏中,并且可以在下拉的通知面板中查看详细信息。用户点击通知后可以触发相应的操作,例如,启动一个 Activity 或打开一个链接。表 5.5 显示了 NotificationManager 的常用方法。

表 5.5 NotificationManager 的常用方法

方 法	功 能
Boolean areBubblesAllowed()	检查应用的通知是否可以显示为气泡浮动在其他应用上方
Boolean areBubblesEnabled()	检查当前用户是否启用了气泡功能
Boolean areNotificationsEnabled()	检查当前应用的通知是否已启用
Boolean areNotificationsPaused()	检查当前应用的通知是否被暂时隐藏
void cancel(int id)	取消指定 ID 的通知
void cancelAll()	取消所有已显示的通知
void createNotificationChannel(NotificationChannel channel)	创建通知通道,指定通知的展示和行为方式
void deleteNotificationChannel(String channelId)	删除指定的通知通道
StatusBarNotification[] getActiveNotifications()	获取当前活跃的通知列表
void notify(int id, Notification notification)	在状态栏中发布一个通知

在应用中使用 NotificationManager 创建通知的步骤如下。

(1) 申请权限。

从 Android 13 开始,必须获取 POST_NOTIFICATIONS 权限,需要在应用启动时或在需要发送通知之前请求此权限,代码如下。

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
    if (checkSelfPermission(Manifest.permission.POST_NOTIFICATIONS) !=
PackageManager.PERMISSION_GRANTED) {
        requestPermissions(new String[]{
            Manifest.permission.POST_NOTIFICATIONS}, 101)});
    }
}
```

（2）获取 NotificationManager 实例。

```
NotificationManager notificationManager = (NotificationManager) getSystemService
(Context.NOTIFICATION_SERVICE);
```

（3）创建通知渠道。

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
    String channelId = "my_channel_id";
    CharSequence name = "My Channel";
    String description = "Channel for My App Notifications";
    int NotificationChannel channel =
        new NotificationChannel(channelId, name,
        channel.setDescription(description);
    //注册渠道
    NotificationManager notificationManager =
        getSystemService(NotificationManager.class);
    notificationManager.createNotificationChannel(channel);
}
```

（4）构建通知。

```
String channelId = "my_channel_id";                    //与创建渠道时使用的 ID 相同
NotificationCompat.Builder builder =
    new NotificationCompat.Builder(this, channelId)
    .setSmallIcon(R.drawable.ic_notification)          //设置小图标
    .setContentTitle("My Notification")                //设置通知标题
    .setContentText("This is the content of the notification.") //设置通知内容
    .setPriority(NotificationCompat.PRIORITY_DEFAULT);         //设置通知优先级
```

（5）发布通知。

```
int notificationId = 1;
notificationManager.notify(notificationId, builder.build());
```

（6）取消通知。

```
notificationManager.cancel(notificationId);
```

5.5　远程服务

5.5.1　什么是远程服务

远程服务是指可以在不同进程之间进行通信的服务。通常情况下，Android 应用程序的组件（如 Activity、Service 等）运行在应用程序自身的进程中，并且默认情况下，这些组件只能被同一个应用内的其他组件访问。然而，通过创建远程服务，可以让一个应用中的服务

被另一个应用访问，从而实现跨应用的进程间通信(IPC)。

远程服务与本地服务存在一些区别。远程服务运行在不同的进程中，如果服务所在的进程崩溃了，它不会影响到调用它的客户端进程。而本地服务则运行在同一进程中，如果服务崩溃了，可能会导致整个应用崩溃。本地服务通常通过直接调用 bindService()来与服务交互，并通过 IBinder 对象直接访问服务的方法。而远程服务则需要通过 AIDL 定义接口，并通过 Binder 机制实现跨进程调用。对于远程服务来说，由于涉及进程间的通信，因此在服务的绑定和解绑过程中，需要处理更多的网络通信相关的问题。此外，当服务进程被杀死或重启时，远程服务的客户端可能需要重新建立连接。

远程服务因为涉及不同应用之间的交互，所以需要更严格的权限管理。例如，可能需要设置权限标签＜permission＞来限制哪些应用可以绑定到服务。远程服务由于运行在不同的进程中，其资源管理也与本地服务不同。例如，当系统内存紧张时，系统可能会优先杀死远程服务所在的进程。

远程服务可以实现多个应用之间的数据共享和功能调用。例如，一个应用可以调用另一个应用提供的远程服务来获取数据或触发特定操作，也可以通过远程服务来保持进程独立性，防止因前台进程被关闭而中断服务。此外，在大型系统中，使用远程服务可以实现应用功能模块化。例如，支付系统、用户管理系统等可以作为独立的服务运行，方便管理和维护。

5.5.2 远程服务中的 AIDL

Android 接口定义语言(AIDL)是一种用于抽象化进程间通信(IPC)的工具。通过在 .aidl 文件中定义接口，各种构建系统会使用 AIDL 二进制文件生成 C++或 Java 绑定，从而实现跨进程使用该接口。AIDL 可以在 Android 中的任何进程之间使用，既可以在平台组件之间使用，也可以在不同应用之间使用。

AIDL 的语法与 Java 语言的接口类似。AIDL 文件中需要指定接口协议以及此协议中使用的各种数据类型和常量。

AIDL 允许指定函数的参数传递方向，可以定义为 in、out 或 inout。in 是默认方向，表示数据从调用方传递给被调用方。out 表示数据从被调用方传递给调用方。inout 是这两种方向的组合。

创建 AIDL 接口的方法：创建一个 .aidl 文件，定义远程服务的接口。在服务中实现 .aidl 文件中定义的接口。在 AndroidManifest.xml 中声明服务。在客户端应用中绑定远程服务。相关步骤与示例代码如下。

(1) 定义 AIDL 接口，创建 IRemoteService.aidl 文件。

```
interface IRemoteService {
    void performTask();
}
```

(2) 创建 RemoteService.java 文件，实现 AIDL 接口。

```
public class RemoteService extends Service {
    private final IRemoteService.Stub binder = new IRemoteService.Stub() {
```

```
        @Override
        public void performTask() throws RemoteException {
            //实现远程任务
        }
    };
    @Override
    public IBinder onBind(Intent intent) {
        return binder;
    }
}
```

(3) 在 AndroidManifest. xml 中声明服务。

```
<service
    android:name=".RemoteService"
    android:exported="true">
    <intent-filter>
        <action android:name="com.example.remoteservice.IRemoteService" />
    </intent-filter>
</service>
```

(4) 在客户端绑定远程服务并使用。

```
public class MainActivity extends AppCompatActivity {
    private IRemoteService remoteService;
    private boolean isBound = false;
    private ServiceConnection connection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            remoteService = IRemoteService.Stub.asInterface(service);
            isBound = true;
            Toast.makeText(MainActivity.this, "服务已绑定",
              Toast.LENGTH_SHORT).show();
        }
        @Override
        public void onServiceDisconnected(ComponentName name) {
            remoteService = null;
            isBound = false;
            Toast.makeText(MainActivity.this, "服务已断开",
              Toast.LENGTH_SHORT).show();
        }
    };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void onBindServiceClicked(View view) {
        Intent intent = new Intent();
```

```
        intent.setClassName("com.example.attendancedemo",
          "com.example.attendancedemo.RemoteService");
        bindService(intent, connection, BIND_AUTO_CREATE);
    }
    public void onUnbindServiceClicked(View view) {
        if (isBound) {
            unbindService(connection);
            isBound = false;
        }
    }
    public void onPerformOperationClicked(View view) {
        if (isBound) {
            try {
                remoteService.performRemoteOperation();
                Toast.makeText(this, "远程操作已执行",
                        Toast.LENGTH_SHORT).show();
            } catch (RemoteException e) {
                e.printStackTrace();
            }
        } else {
            Toast.makeText(this, "服务未绑定", Toast.LENGTH_SHORT).show();
        }
    }
}
```

5.5.3 远程服务示例

【例 5.5】 远程序服务调用示例。RemoteServiceDemo 是调用远程服务将本地的两个随机数进行相加。

(1) 创建新的远程服务模块 RemoteServiceDemo。

(2) 首先需要在 remoteservicedemo 模块的 build.gradle 文件中添加 aidl 支持。

在 android 块中添加 buildFeatures 配置。

```
android {
    ...
    buildFeatures {
        aidl = true
    }
    ...
}
```

配置好后需要进行 Gradle 同步,否则后续无法添加 AIDL 文件。

(3) 创建布局文件 remoteservicedemo\activity_main.xml。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <Button
        android:id="@+id/bindServiceButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="绑定远程服务" />
    <Button
        android:id="@+id/addNumbersButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="随机数相加" />
    <TextView
        android:id="@+id/resultTextView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:text="结果 = " />
</LinearLayout>
```

(4) 创建 IRemoteService.aidl 文件,右击模块名,在 Android Studio 中选择 File→new→AIDL→AIDL File,这是声明远程服务内公共函数的文件,只需要声明,不需要在这里实现。

```
程序代码:remoteservicedemo\IRemoteService.aidl
package com.example.remoteservicedemo;
interface IRemoteService {
    //程序中定义 addNumbers()方法,接收两个参数并求和
    double addNumbers(double a, double b);
}
```

Build 编译项目,生成 IRemoteService 类文件。

(5) 创建 RemoteService.java 文件,实现 AIDL 接口。

```
程序代码:remoteservicedemo\RemoteService.java
package com.example.remoteservicedemo;
public class RemoteService extends Service {
    //实现 AIDL 接口
    private final IRemoteService.Stub binder = new IRemoteService.Stub() {
        @Override
        public double addNumbers(double a, double b) throws RemoteException {
            Log.d("RemoteService", "addNumbers");
            return a + b;
        }
    };
    @Nullable
    @Override
```

```
    public IBinder onBind(Intent intent) {
        Log.d("RemoteService", "onBind");
        return binder;
    }
}
```

（6）创建 MainActivity，程序界面与用户交互生成随机数并调用远程服务。

```
程序代码：remoteservicedemo\MainActivity.java
package com.example.remoteservicedemo;
public class MainActivity extends AppCompatActivity {
    //远程服务接口
    private IRemoteService remoteService;
    //服务是否绑定的标志
    private boolean isBound = false;
    //服务连接对象
    private ServiceConnection serviceConnection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            Log.d("MainActivity","onServiceConnected");
            //获取远程服务接口
            remoteService = IRemoteService.Stub.asInterface(service);
            isBound = true;
        }
        @Override
        public void onServiceDisconnected(ComponentName name) {
            Log.d("MainActivity","onServiceDisconnected");
            remoteService = null;
            isBound = false;
        }
    };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //启用 EdgeToEdge
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);
        //设置窗口插入监听器
         ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main),
(v, insets) -> {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.
systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right,
systemBars.bottom);
            return insets;
        });
        //获取按钮和文本视图
        Button bindServiceButton = findViewById(R.id.bindServiceButton);
        Button addNumbersButton = findViewById(R.id.addNumbersButton);
```

```
        TextView resultTextView = findViewById(R.id.resultTextView);
        //绑定服务按钮点击事件
        bindServiceButton.setOnClickListener(v -> {
            Intent intent = new Intent();
             intent.setComponent(new ComponentName("com.example.remoteservicedemo",
"com.example.remoteservicedemo.RemoteService"));
            bindService(intent, serviceConnection, Context.BIND_AUTO_CREATE);
        });
        //加法按钮点击事件
        addNumbersButton.setOnClickListener(v -> {
            if (isBound && remoteService != null) {
                try {
                    //调用远程服务的 addNumbers()方法
                    double result = remoteService.addNumbers(random(), random());
                    resultTextView.setText("结果: " + result);
                } catch (RemoteException e) {
                    e.printStackTrace();
                }
            } else {
                resultTextView.setText("无法调用远程服务!");
                Log.e("MainActivity","无法调用远程服务!");
            }
        });
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        //解绑服务
        if (isBound) {
            unbindService(serviceConnection);
            isBound = false;
        }
    }
}
```

(7) 在配置文件 AndroidManifest.xml 中添加服务的调用。

```
<service android:name=".RemoteService" android:exported="true">
    <intent-filter>
        <action android:name="com.example.remoteservicedemo.IRemoteService" />
    </intent-filter>
</service>
```

程序运行结果如图 5.11 所示,绑定远程服务,生成两个随机数并通过远程服务进行运算。

图 5.11　远程服务运行结果

习题

一、单项选择题

1. 下列选项中，属于可以在后台持续运行的组件是(　　)。
   A. Activity　　B. ContentProvider
   C. Service　　D. Intent
2. 关于 Service 的描述中，哪一项是错误的？(　　)
   A. Service 是 Android 的 4 大组件之一
   B. 没有用户界面
   C. 在 Java 代码中可以动态注册服务
   D. Service 依赖于 Activity，当 Activity 销毁时，Service 也被销毁
3. 下列关于 Service 服务的描述中，错误的是(　　)。
   A. Service 是没有用户可见的界面，不能与用户交互
   B. Service 可以通过 Context. startService()来启动
   C. Service 可以通过 Context. bindService()来启动
   D. Service 无须在清单文件中进行配置。
4. 下列关于 Service 生命周期的说法，正确的是(　　)。
   A. 服务的生命周期和 Activity 一样
   B. 服务的创建会执行 onCreate()
   C. 启动时 onCreate()→onStart()→onResume()
   D. 通过 startService()方法开启服务，首先会调用 onCreate()和 onStart()方法
5. 定义一个非绑定服务时，需要重写 Service 的哪些方法？(　　)
   A. onCreate()→onResume()→onDestroy()
   B. onCreate()→onStartCommand()→onDestroy()
   C. onCreate()→onResume()→onPause→onDestroy()
   D. onCreate()→onResume()→onStart()→onDestroy()
6. 下列关于 Service 的方法描述，错误的是(　　)。
   A. onCreate()表示第一次创建服务时执行的方法
   B. 调用 startService()方法启动服务时执行的方法是 onStartCommand()
   C. 调用 bindService()方法启动服务时执行的方法是 onBind()
   D. 调用 startService ()方法断开服务绑定时执行的方法是 onUnbind()

7. 通过 startService()启动服务时，以下哪项是正确的？(　　)

A. 服务停止时会调用 onStop()

B. 服务开启后只能关机后才能关闭服务

C. 服务不需要在清单文件里注册

D. startService()方法开启服务，服务一旦被开启，就会在后台长期运行

8. 如果通过 bindService()方法开启服务，那么服务的生命周期是(　　)。

A. onCreate()→onstart()→onBind()→onDestroy()

B. onCreate()→onBind()→onDestroy()

C. onCreate()→onBind()→onUnBind()→onDestroy()

D. onCreate()→onStart()→onBind ()→onUnBind()→onDestroy()

9. 下面关于 bindService()方法启动服务的描述，正确的是(　　)。

A. 服务会长期在后台运行

B. 启动服务的组件与服务之间没有关联

C. 可以通过 stopService()方法停止该服务

D. 可以通过 unbindService()方法停止该服务

10. 下列选项中，当使用 bindService()启动服务时，停止服务调用的方法是(　　)。

A. stopSelf()　　B. stopService()

C. unbindService()　　D. finish()

二、判断题

1. 通过绑定方式启动服务后，服务与调用者没有关系(　　)。

2. 服务的界面可以设置得很美观。(　　)

3. 通过绑定方式启动服务后，当界面不可见时服务就会被关闭(　　)。

4. 在服务中可以处理长时间的耗时操作。(　　)

5. 服务不是 Android 中的 4 大组件，因此不需要在清单文件中注册。(　　)

6. Service 服务是运行在子线程中的。(　　)

7. 使用服务的通信方式进行通信时，必须保证服务是以绑定的方式开启的，否则无法通信。(　　)

8. 一个组件只能绑定一个服务。(　　)

三、填空题

1. 在创建服务时，必须要继承_____类。

2. 绑定服务时，必须要实现服务的_____方法。

3. 在清单文件中，注册服务时应该使用的标签是_____。

4. 服务的开启方式有两种，分别是_____和_____。

5. 在进行远程服务通信时，需要使用_____接口。

6. 如果想要停止 bindService()方法启动的服务，需要调用_____方法。

7. Android 系统的服务的通信方式分为_____和_____。

四、简答题

1. 简述 Service 后台服务与绑定服务的区别。

2. 简述 Service 的生命周期中各个方法的用途。

3. 什么是远程服务？