什么是服务？

Service是一个应用程序组件，它能够在后台执行一些耗时较长的操作，并且不提供用户界面。服务能被其它应用程序的组件启动，即使用户切换到另外的应用时还能保持后台运行。此外，应用程序组件还能与服务绑定，并与服务进行交互，甚至能进行进程间通信（IPC）。 比如，服务可以处理网络传输、音乐播放、执行文件I/O、或者与content provider进行交互，所有这些都是后台进行的。

Service 与 Thread 的区别

服务仅仅是一个组件，即使用户不再与你的应用程序发生交互，它仍然能在后台运行。因此，应该只在需要时才创建一个服务。

如果你需要在主线程之外执行一些工作，但仅当用户与你的应用程序交互时才会用到，那你应该创建一个新的线程而不是创建服务。比如，如果你需要播放一些音乐，但只是当你的activity在运行时才需要播放，你可以在onCreate()中创建一个线程，在onStart()中开始运行，然后在onStop()中终止运行。还可以考虑使用AsyncTask或HandlerThread来取代传统的Thread类。

由于无法在不同的 Activity 中对同一 Thread 进行控制，这个时候就要考虑用服务实现。如果你使用了服务，它默认就运行于应用程序的主线程中。因此，如果服务执行密集计算或者阻塞操作，你仍然应该在服务中创建一个新的线程来完成（避免ANR）。

服务的分类

按运行分类

- 前台服务

前台服务是指那些经常会被用户关注的服务，因此内存过低时它不会成为被杀的对象。 前台服务必须提供一个状态栏通知，并会置于“正在进行的”（“Ongoing”）组之下。这意味着只有在服务被终止或从前台移除之后，此通知才能被解除。 例如，用服务来播放音乐的播放器就应该运行在前台，因为用户会清楚地知晓它的运行情况。 状态栏通知可能会标明当前播放的歌曲，并允许用户启动一个activity来与播放器进行交互。

要把你的服务请求为前台运行，可以调用startForeground()方法。此方法有两个参数：唯一标识通知的整数值、状态栏通知Notification对象。例如：

```
Notification notification = new Notification(R.drawable.icon, getText(R.string.ticker_text),System.cu

Intent notificationIntent = new Intent(this,ExampleActivity.class);

PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);

notification.setLatestEventInfo(this, getText(R.string.notification_title),
        getText(R.string.notification_message), pendingIntent);

startForeground(ONGOING_NOTIFICATION, notification);
```

要从前台移除服务，请调用stopForeground()方法，这个方法接受个布尔参数，表示是否同时移除状态栏通知。此方法不会终止服务。不过，如果服务在前台运行时被你终止了，那么通知也会同时被移除。

- 后台服务

按使用分类

- 本地服务

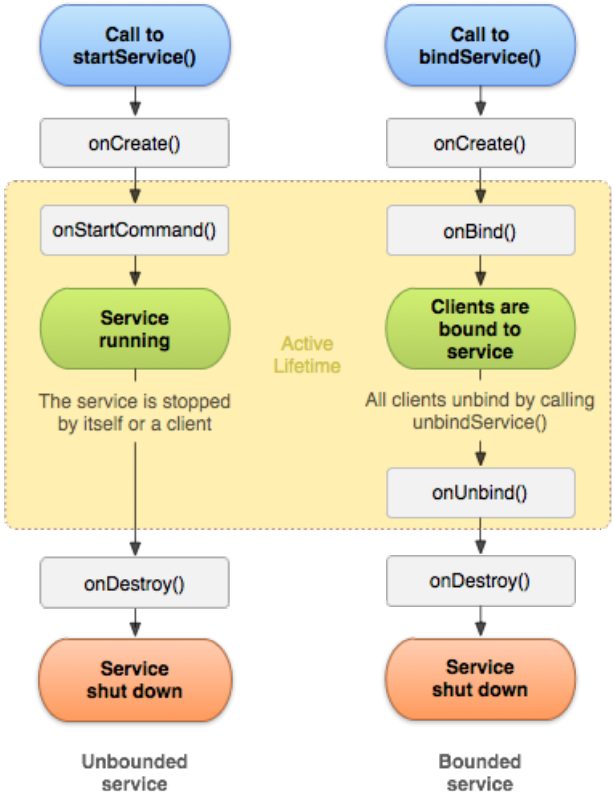
用于应用程序内部，实现一些耗时任务，并不占用应用程序比如Activity所属线程，而是单开线程后台执行。 调用Context.startService()启动，调用Context.stopService()结束。在内部可以调用Service.stopSelf() 或 Service.stopSelfResult()来自己停止。

- 远程服务

用于Android系统内部的应用程序之间，可被其他应用程序复用，比如天气预报服务，其他应用程序不需要再写这样的服务，调用已有的即可。可以定义接口并把接口暴露出来，以便其他应用进行操作。客户端建立到服务对象的连接，并通过那个连接来调用服务。调

用Context.bindService()方法建立连接，并启动，以调用 Context.unbindService()关闭连接。多个客户端可以绑定至同一个服务。如果服务此时还没有加载，bindService()会先加载它。

Service生命周期



Service生命周期方法：

```
public class ExampleService extends Service {
    int mStartMode;       // 标识服务被杀死后的处理方式
    IBinder mBinder;      // 用于客户端绑定的接口
    boolean mAllowRebind; // 标识是否使用onRebind

    @Override
    public void onCreate() {
        // 服务正被创建
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        // 服务正在启动，由startService()调用引发
        return mStartMode;
    }
    @Override
    public IBinder onBind(Intent intent) {
        // 客户端用bindService()绑定服务
        return mBinder;
    }
    @Override
    public boolean onUnbind(Intent intent) {
        // 所有的客户端都用unbindService()解除了绑定
        return mAllowRebind;
    }
    @Override
    public void onRebind(Intent intent) {
        // 某客户端正用bindService()绑定到服务，
        // 而onUnbind()已经被调用过了
    }
    @Override
    public void onDestroy() {
        // 服务用不上了，将被销毁
    }
}
```

> 请注意onStartCommand()方法必须返回一个整数。这个整数是描述系统在杀死服务之后应该如何继续运行。onStartCommand()的返回值必须是以下常量之一：

> START_NOT_STICKY　如果系统在onStartCommand()返回后杀死了服务，则不会重建服务了，除非还存在未发送的intent。当服务不再是必需的，并且应用程序能够简单地重启那些未完成的工作时，这是避免服务运行的最安全的选项。
> START_STICKY 如果系统在onStartCommand()返回后杀死了服务，则将重建服务并调用onStartCommand()，但不会再次送入上一个intent， 而是用null intent来调用onStartCommand() 。除非还有启动服务的intent未发送完，那么这些剩下的intent会继续发送。 这适用于媒体播放器（或类似服务），它们不执行命令，但需要一直运行并随时待命。
>
> START_REDELIVER_INTENT 如果系统在onStartCommand()返回后杀死了服务，则将重建服务并用上一个已送过的intent调用onStartCommand()。任何未发送完的intent也都会依次送入。这适用于那些需要立即恢复工作的活跃服务，比如下载文件。

服务的生命周期与activity的非常类似。不过，更重要的是你需密切关注服务的创建和销毁环节，因为后台运行的服务是不会引起用户注意的。

服务的生命周期——从创建到销毁——可以有两种路径：

- 一个started服务

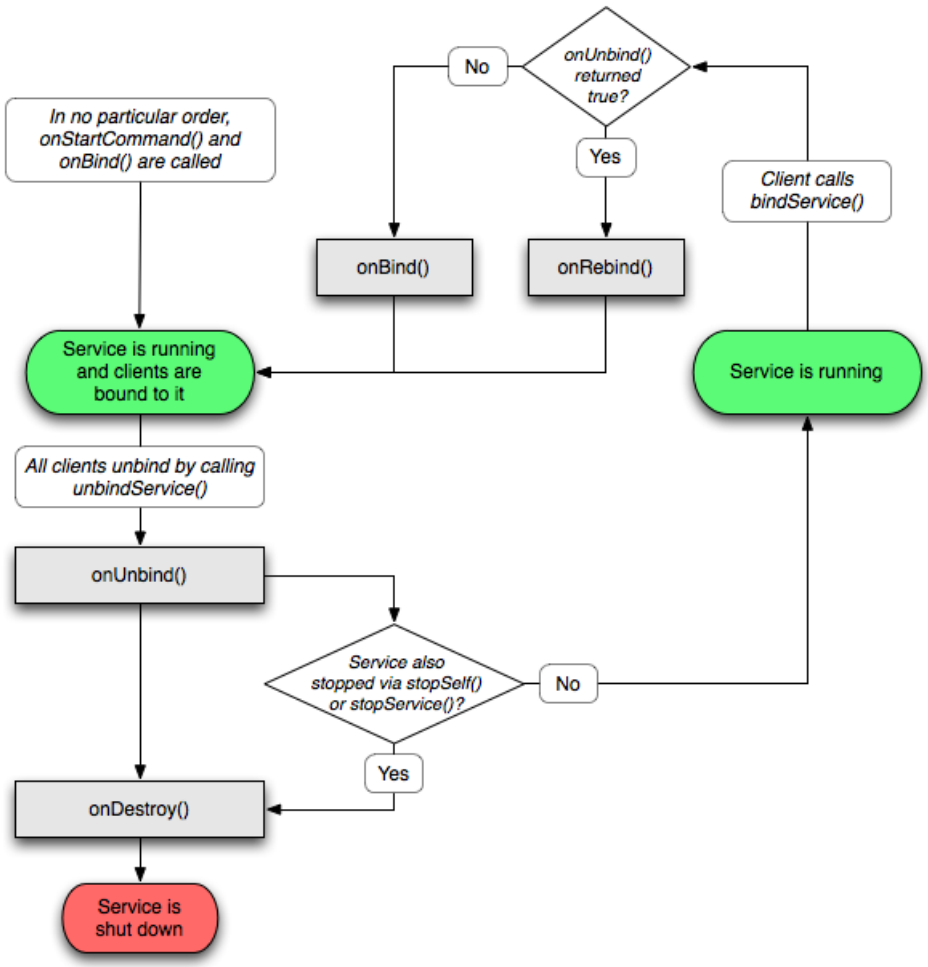
这类服务由其它组件调用startService()来创建。然后保持运行，且必须通过调用stopSelf()自行终止。其它组件也可通过调用stopService() 终止这类服务。服务终止后，系统会把它销毁。

如果一个Service被startService 方法多次启动，那么onCreate方法只会调用一次，onStart将会被调用多次（对应调用startService的次数），并且系统只会创建Service的一个实例（因此你应该知道只需要一次stopService调用）。该Service将会一直在后台运行，而不管对应程序的Activity是否在运行，直到被调用stopService，或自身的stopSelf方法。当然如果系统资源不足，android系统也可能结束服务。

- 一个bound服务

服务由其它组件（客户端）调用bindService()来创建。然后客户端通过一个IBinder接口与服务进行通信。客户端可以通过调用unbindService()来关闭联接。多个客户端可以绑定到同一个服务上，当所有的客户端都解除绑定后，系统会销毁服务。（服务不需要自行终止。）

如果一个Service被某个Activity 调用 Context.bindService 方法绑定启动，不管调用 bindService 调用几次，onCreate方法都只会调用一次，同时onStart方法始终不会被调用。当连接建立之后，Service将会一直运行，除非调用Context.unbindService 断开连接或者之前调用bindService 的 Context 不存在了（如Activity被finish的时候），系统将会自动停止Service，对应onDestroy将被调用。



这两条路径并不是完全隔离的。也就是说，你可以绑定到一个已经用startService()启动的服务上。例如，一个后台音乐服务可以通过调用startService()来启动，传入一个指明所需播放音乐的 Intent。 之后，用户也许需要用播放器进行一些控制，或者需要查看当前歌曲的信息，这时一个activity可以通过调用bindService()与此服务绑定。在类似这种情况下，stopService()或stopSelf()不会真的终止服务，除非所有的客户端都解除了绑定。

> 当在旋转手机屏幕的时候，当手机屏幕在“横”“竖”变换时，此时如果你的 Activity 如果会自动旋转的话，旋转其实是 Activity 的重新创建，因此旋转之前的使用 bindService 建立的连接便会断开（Context 不存在了）。

在manifest中声明服务

无论是什么类型的服务都必须在manifest中申明，格式如下：

```
<manifest ... >
  ...
  <application ... >
      <service android:name=".ExampleService" />
      ...
  </application>
</manifest>
```

Service 元素的属性有：

> android:name ---------- 服务类名
>
> android:label ---------- 服务的名字，如果此项不设置，那么默认显示的服务名则为类名
>
> android:icon ---------- 服务的图标
>
> android:permission ------- 申明此服务的权限，这意味着只有提供了该权限的应用才能控制或连接此服务
>
> android:process ---------- 表示该服务是否运行在另外一个进程，如果设置了此项，那么将会在包名后面加上这段字符串表示另一进程的名字
>
> android:enabled ---------- 如果此项设置为 true，那么 Service 将会默认被系统启动，不设置默认此项为 false
>
> android:exported --------- 表示该服务是否能够被其他应用程序所控制或连接，不设置默认此项为 false

android:name是唯一必需的属性——它定义了服务的类名。与activity一样，服务可以定义intent过滤器，使得其它组件能用隐式intent来调用服务。如果你想让服务只能内部使用（其它应用程序无法调用），那么就不必（也不应该）提供任何intent过滤器。 此外，如果包含了android:exported属性并且设置为"false"， 就可以确保该服务是你应用程序的私有服务。即使服务提供了intent过滤器，本属性依然生效。

startService 启动服务

从activity或其它应用程序组件中可以启动一个服务，调用startService()并传入一个Intent（指定所需启动的服务）即可。

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

服务类：

```
public class MyService extends Service {

      /**
     * onBind 是 Service 的虚方法，因此我们不得不实现它。
     * 返回 null，表示客服端不能建立到此服务的连接。
     */
    @Override
    public IBinder onBind(Intent intent) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();
    }

    @Override
 public int onStartCommand(Intent intent, int flags, int startId)       {
    //接受传递过来的intent的数据
     return START_STICKY;
    };

    @Override
    public void onDestroy() {
        super.onDestroy();
    }

}
```

一个started服务必须自行管理生命周期。也就是说，系统不会终止或销毁这类服务，除非必须恢复系统内存并且服务返回后一直维持运行。 因此，服务必须通过调用stopSelf()自行终止，或者其它组件可通过调用stopService()来终止它。

bindService 启动服务

当应用程序中的activity或其它组件需要与服务进行交互，或者应用程序的某些功能需要暴露给其它应用程序时，你应该创建一个bound服务，并通过进程间通信（IPC）来完成。

方法如下：

```
Intent intent=new Intent(this,BindService.class);
bindService(intent, ServiceConnection conn, int flags)
```

> 注意bindService是Context中的方法，当没有Context时传入即可。

在进行服务绑定的时，其flags有：

- Context.BIND_AUTO_CREATE

表示收到绑定请求的时候，如果服务尚未创建，则即刻创建，在系统内存不足需要先摧毁优先级组件来释放内存，且只有驻留该服务的进程成为被摧毁对象时，服务才被摧毁

- Context.BIND_DEBUG_UNBIND

通常用于调试场景中判断绑定的服务是否正确，但容易引起内存泄漏，因此非调试目的的时候不建议使用

- Context.BIND_NOT_FOREGROUND

表示系统将阻止驻留该服务的进程具有前台优先级，仅在后台运行。

服务类：

```
public class BindService extends Service {

     // 实例化MyBinder得到mybinder对象;
    private final MyBinder binder = new MyBinder();

      /**
     * 返回Binder对象。
     */
    @Override
    public IBinder onBind(Intent intent) {
        // TODO Auto-generated method stub
        return binder;
    }

     /**
      * 新建内部类MyBinder, 继承自Binder(Binder实现IBinder接口),
      * MyBinder提供方法返回BindService实例。
      */
   public class MyBinder extends Binder{

        public BindService getService(){
            return BindService.this;
        }
    }


    @Override
    public boolean onUnbind(Intent intent) {
        // TODO Auto-generated method stub
        return super.onUnbind(intent);
    }
}
```

启动服务的activity代码：

```
public class MainActivity extends Activity {

    /** 是否绑定 */
    boolean mIsBound = false;
    BindService mBoundService;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        doBindService();
    }
    /**
     * 实例化ServiceConnection接口的实现类,用于监听服务的状态
     */
    private ServiceConnection conn = new ServiceConnection() {

        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            BindService mBoundService = ((BindService.MyBinder) service).getService();

        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
            mBoundService = null;

        }
    };

    /** 绑定服务 */
    public void doBindService() {
        bindService(new Intent(MainActivity.this, BindService.class), conn,Context.BIND_AUTO_CREATE);
        mIsBound = true;
    }

    /** 解除绑定服务 */
    public void doUnbindService() {
        if (mIsBound) {
            // Detach our existing connection.
            unbindService(conn);
            mIsBound = false;
        }
    }

    @Override
    protected void onDestroy() {
        // TODO Auto-generated method stub
        super.onDestroy();

        doUnbindService();
    }
}
```

> 注意在AndroidMainfest.xml中对Service进行显式声明

判断Service是否正在运行：

```
private boolean isServiceRunning() {
    ActivityManager manager = (ActivityManager) getSystemService(ACTIVITY_SERVICE);

   {
        if ("com.example.demo.BindService".equals(service.service.getClassName()))     {
            return true;
        }
    }
    return false;
}
```