Android 之 Service 的细枝末节，有你想了解的！

阅读 1306 收藏 87 2016-12-19
原文链接：blog.csdn.net

转载注明出处：blog.csdn.net/xiaohanluo/…

本人在参加CSDN博客之星评选，如果觉得文章对你有帮助，麻烦帮我投一票，投票链接，谢谢。

1. 简介

与前一篇Android之Activity的细枝末节是同一系列的文章，是自己在学习和研发过程中，对Service的一些知识点的总结，汇总得到这篇文章。

这篇文章会从Service的一些小知识点，延伸到**Android**中几种常用进程间通信方法。

2. 进程

Service是一种不提供用户交互页面但是可以在后台长时间运行的组件，可以通过

应用并不是同一个进程。

四大组件都支持 `android:process=":remote"` 这个属性。

因为Service可以运行在不同的进程，这里说一下Android中几种进程的优先级，当系统内存不足时候，系统会从优先级低的进程开始回收，下面根据优先级由高到低列出Android中几种进程。

- 前台进程，当前用户操作所需要的进程
    - 用户正在交互的Activity（Activity执行了onResume方法）
    - 与正在交互的Activity绑定的Service
    - 设置为前台权限的Service（Service调用startForeground()方法）
    - 正在执行某些生命周期回调的Service，onCreate()、onStart()、onDestroy()
    - 正在执行onReceive()的BroadcastReceiver

  这种进程基本不会被回收，只有当内存不足以支持前台进程同时运行时候，系统才回回收它们，主要关注前三个。

- 可见进程，没有与用户交互所必须的组件，但是在屏幕上仍然可见其内容的进程
    - 调用了onPause()方法但仍对用户可见的Activity
    - 与上面这种Activity绑定的Service

- 服务进程，使用startService()启动的Service且不属于上面两种类别进程的进程，虽然这个进程与用户交互没有直接关系，但是一般会在后台执行一些耗时操作，所以，只有当内存不足以维持所有前台进程和可见进程同时运行，系统才回回收这个类别的进程。

- 后台进程，对用户不可见的Activity进程，已调用了onStop()方法的Activity

- 空进程，不包含任何活动应用组件的进程，保留这种进程唯一目的是作为缓存，引用组件下次启动时间。通常系统会最优先回收这类进程。

3. Service配置

和其他组件(Activity/ContentProvider/BroadcastReceiver)一样，Service需要在Androidmanifest.xml中声明。

```
...

    ...
```

Service是运行在主线程中的，如果有什么耗时的操作，建议新建子线程去处理，避免阻塞主线程，降低ANR的风险。

在另外一篇文章中Intent以及IntentFilter详解提到过，为了确保应用的安全，不要为Service设置intent-filter，因为如果使用隐式Intent去启动Service时候，手机里面那么多应用，并不能确定哪一个Service响应了这个Intent，所以在项目中尽量使用显式Intent去启动Service。在Android 5.0(API LEVEL 21)版本后的，如果传入隐式Intent去调用 `bindService()` 方法，系统会抛出异常。

可以通过设置 `android:exported=false` 来确保这个Service仅能在本应用中使用。

4. 服务启动方式

服务可以由其他组件启动，而且如果用户切换到其他应用，这个服务可能会继续在后台执行。到目前为止，Android中Service总共有三种启动方式。

- Scheduled，可定时执行的Service，是Android 5.0（API LEVEL 21）版本中新添加的一个Service，名为JobService，继承Service类，使用JobScheduler类调度它并且设置JobService运行的一些配置。具体文档可以参考JobScheduler，如果你的应用最低支持版本是21，官方建议使用JobService。

Service B，过了会儿，Activity A执行onDestroy()被销毁了，如果Service B任务没有执行完毕，它仍然会在后台执行。这种启动方式启动的Service需要主动调用 `StopService()` 停止服务。

- Bound，通过 `bindService()` 启动的Service。通过这种方式启动Service时候，会返回一个客户端交互接口，用户可以通过这个接口与服务进行交互，如果这个服务是在另一个进程中，那么就实现了进程间通信，也就是Messenger和AIDL，这个会是下篇文章的重点。多个组件可以同时绑定同一个Service，如果所有的组件都调用 `unbindService()` 解绑后，Service会被销毁。

startService和bindService可以同时使用

5. 主要方法

Service是一个抽象类，使用需要我们去实现它的抽象方法 `onBind()` ，Service有且仅有这一个抽象方法，还有一些其他的生命周期回调方法需要复写帮助我们实现具体的功能。

- onCreate()，在创建服务时候，可以在这个方法中执行一些的初始化操作，它在 `onStartCommand()` 和 `onBind()` 之前被调用。如果服务已经存在，调用 `startService()` 启动服务时候这个方法不会调用，只会调用 `onStartCommand()` 方法。
- onStartCommand()，其他组件通过 `startService()` 启动服务时候会回调这个方法，这个方法执行后，服务会启动被在后台运行，需要调用 `stopSelf()` 或者 `stopService()` 停止服务。
- onBind()，其他组件通过 `bindService()` 绑定服务时候会回调的方法，这是Service的一个抽象方法，如果客户端需要与服务交互，需要在这个方法中返回一个 `IBinder` 实现类实例化对象，如果不想其他客户端与服务绑定，直接返回null。
- onDestroy()，当服务不在还是用且即将被销毁时，会回调这个方法，可以在这个方法中做一些释放资源操作，这是服务生命周期的最后一个回调。

如果组件仅通过 `startService()` 启动服务，不论服务是否已经启动，都会回调 `onStartCommand()` 方法，而且服务会一直运行，需要调用 `stopSelf` 和 `stopService` 方法关

的客户端全部取消绑定 `unbindService` ，系统才会销毁该服务。

多次启动同一个服务，只有在服务初次启动时候会回调 `onCreate` 方法，但是每次都会回调 `onStartCommand` ，可以利用这个向服务传递一些信息。

onStartCommand()的回调是在**UI**主线程，如果有什么耗时的操作，建议新启线程去处理。

6. 启动和关闭服务

启动服务：

- `JobScheduler.schedule()`
- `startService(Intent)`
- `bindService(Intent service, ServiceConnection conn, int flags)`

关闭服务：

- `JobScheduler.cancel()` 或者 `JobScheduler.cancelAll()` ，对应 `JobScheduler.schedule()`
- Service自身的 `stopSelf()` 方法，组件的 `stopService(Intent)` 方法，对应 `startService` 启动方法
- `unbindService(ServiceConnection conn)` ，对应 `bindService`

示例：

```
// 启动服务
Intent intent = new Intent(this, DemoService.class);
startService(intent);

// 停止服务
```

```
ServiceConnection mConnection = ServiceConnection() { ... };
Intent intent = new Intent(this, DemoService.class);
bindService(intent, mConnection, Context.BIND_AUTO_CREATE);

// 解除绑定
unbindService(mConnection);
```

绑定服务 `bindService()` 第三个参数数值：

- 0，如果不想设置任何值，就设置成0
- `Context.BIND_AUTO_CREATE` ，绑定服务时候，如果服务尚未创建，服务会自动创建，在API LEVEL 14以前的版本不支持这个标志，使用 `Context.BIND_WAIVE_PRIORITY` 可以达到同样效果
- `Context.BIND_DEBUG_UNBIND` ，通常用于Debug，在 `unbindService` 时候，会将服务信息保存并打印出来，这个标记很容易造成内存泄漏。
- `Context.BIND_NOT_FOREGROUND` ，不会将被绑定的服务提升到前台优先级，但是这个服务也至少会和客户端在内存中优先级是相同的。
- `Context.BIND_ABOVE_CLIENT` ，设置服务的进程优先级高于客户端的优先级，只有当需要服务晚于客户端被销毁这种情况才这样设置。
- `Context.BIND_ALLOW_OOM_MANAGEMENT` ，保持服务受默认的服务管理器管理，当内存不足时候，会销毁服务
- `Context.BIND_WAIVE_PRIORITY` ，不会影响服务的进程优先级，像通用的应用进程一样将服务放在一个LRU表中
- `Context.BIND_IMPORTANT` ，标识服务对客户端是非常重要的，会将服务提升至前台进程优先级，通常情况下，即时客户端是前台优先级，服务最多也只能被提升至可见进程优先级，
- `BIND_ADJUST_WITH_ACTIVITY` ，如果客户端是Activity，服务优先级的提高取决于Activity的进程优先级，使用这个标识后，会无视其他标识。

onStartCommand() 方法有一个int的返回值，这个返回值标识服务关闭后系统的后续操作。

返回值有以下几种：

- `Service.START_STICKY` ，启动后的服务被杀死，系统会自动重建服务并调用 `onStartCommand()` ，但是不会传入最后一个Intent(Service可能多次执行onStartCommand)，会传入一个空的Intent，使用这个标记要注意对Intent的判空处理。这个标记适用于太依靠外界数据Intent，在特定的时间，有明确的启动和关闭的服务，例如后台运行的音乐播放。
- `Service.START_NOT_STICKY` ，启动后的服务被杀死，系统不会自动重新创建服务。这个标记是最安全的，适用于依赖外界数据Intent的服务，需要完全执行的服务。
- `Service.START_REDELIVER_INTENT` ，启动后的服务被杀死，系统会重新创建服务并调用 `onStartCommand()` ，同时会传入最后一个Intent。这个标记适用于可恢复继续执行的任务，比如说下载文件。
- `Service.START_STICKY_COMPATIBILITY` ，启动后的服务被杀死，不能保证系统一定会重新创建Service。

8. Service生命周期

Service生命周期（从创建到销毁）跟它被启动的方式有关系，这里只介绍 `startService` 和 `bindService` 两种启动方法时候Service的生命周期。

- `startService` 启动方式，其他组件用这种方式启动服务，服务会在后台一直运行，只有服务调用本身的 `stopSelf()` 方法或者其他组件调用 `stopService()` 才能停止服务。
- `bindService` 启动方式，其他组件用这种方法绑定服务，服务通过IBinder与客户端通信，客户端通过 `unbindService` 接触对服务的绑定，当没有客户端绑定到服务，服务会被系统销毁。

这两种生命周期不是独立的，组件可以同时用 `startService` 启动服务同时用 `bindSer`

才会被销毁。

图-1Service生命周期图

左图是使用 `startService()` 所创建的服务的生命周期，右图是使用 `bindService()` 所创建的服务的生命周期。

9. 在前台运行服务

服务可以通过startForeground来使服务变成前台优先级。

```
public final void startForeground(int id, Notification notification) {
    try {
        mActivityManager.setServiceForeground(
                new ComponentName(this, mClassName), mToken, id,
                notification, true);
    } catch (RemoteException ex) {
    }
}
```

第一个参数用于标识你应用中唯一的通知标识**id**，不能设为0，最终会传入 `NotificationManager.notify(int id, Notification notification)` 取消通知需要用到，第二个参数是通知具体内容。

```
// 设置Notification属性
Notification notification = new Notification(R.drawable.icon, getText(R.string.ticker
Intent notificationIntent = new Intent(this, ExampleActivity.class);
PendingIntent pendingIntent = PendingIntent.getActivity(this, 0, notificationIntent,
notification.setLatestEventInfo(this, getText(R.string.notification_title), getText(R
startForeground(ONGOING_NOTIFICATION_ID, notification);
```

要将服务从前台移除，需要调用stopForeground(boolean removeNotification)，参数是一个布尔值，用来标识服务从前台服务移除时候，是否需要移除状态栏的通知。如果服务在前台运行时候被停止，状态栏的通知也会被移除。

10. 与服务通信

10.1 广播

不多说，万能的通信。

10.2 本地数据共享

不多说，万能的通信，例如ContentProvider/SharePreference等等。

10.3 startService()

使用这个方法启动的服务，再次调用 `startService()` 传入Intent即可与服务通信，因为这种方式启动的服务在完整的生命周期内 `onCreate()` 只会执行一次，而 `onStartCommand()` 会执行多次，我们再次调用 `startService()` 时候，可以在 `oonStartCommand()` 去处理。

10.4 bindService()

使用这种方法启动的服务，组件有三种与服务通信的方式。

- Service中实现IBinder

下一篇文章具体介绍Messenger、AIDL，因为它们是属于Android进程间通信。

如果一个服务Service只需要在本应用的进程中使用，不提供给其他进程，推荐使用第一种方法。

使用示例：

Service：

```
/**
 * 本地服务
 *

 * 和启动应用属于同一进程
 */
public class LocalService extends Service {
    /**
     * 自定的IBinder
     */
    private final IBinder mBinder = new LocalBinder();

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

    /**
     * 提供给客户端的方法
     *
     * @return
     */
    public String getServiceInfo() {
        return this.getPackageName() + " " + this.getClass().getSimpleName();
    }
```

```
     */
    public class LocalBinder extends Binder {
        public LocalService getService() {
            return LocalService.this;
        }
    }
}
```

Activity：

```
/**
 * 绑定本地服务的组件
 *
 * Created by KyoWang.
 */
public class BindLocalServiceActivity extends AppCompatActivity implements View.OnCli

    private Button mShowServiceNameBtn;

    private LocalService mService;

    private boolean mBound = false;

    public ServiceConnection mConnection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            LocalService.LocalBinder binder = (LocalService.LocalBinder) service;
            mService = binder.getService();
            mBound = true;
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
            mBound = false;
```

```
@Override
protected void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.a_bind_local_service);
    mShowServiceNameBtn = (Button) findViewById(R.id.bind_local_service_btn);
    mShowServiceNameBtn.setOnClickListener(this);
}

@Override
protected void onStart() {
    super.onStart();
    Intent intent = new Intent(this, LocalService.class);
    bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
}

@Override
protected void onStop() {
    super.onStop();
    if(mBound) {
        unbindService(mConnection);
        mBound = false;
    }
}

@Override
public void onClick(View v) {
    int id = v.getId();
    if(id == R.id.bind_local_service_btn) {
        if(mBound) {
            String info = mService.getServiceInfo();
            Toast.makeText(BindLocalServiceActivity.this, info, Toast.LENGTH_SHOR
        }
    }
}
```

11. 服务长存后台

关于网上通用的提升服务优先级以保证服务长存后台，即保证服务不轻易被系统杀死的方法有以下几种。

- 设置 `android:persistent="true"` ，这是application的一个属性，官方都不建议使用。

```
Whether or not the application should remain running at all times,
"true" if it should, and "false" if not.
The default value is "false".
Applications should not normally set this flag;
persistence mode is intended only for certain system applications.
```

- 设置 `android:priority` 优先级，这个并不是Service的属性。这个属性是在 `intent-filter` 中设置的。官方解释，这个属性只对活动和广播有用，而且这个是接受Intent的优先级，并不是在内存中的优先级，呵呵。

```
android:priority
The priority that should be given to the parent component with regard to
handling intents of the type described by the filter.
This attribute has meaning for both activities and broadcast receivers。
```

- 在Service的 `onDestroy` 中发送广播，然后重启服务，就目前我知道的，会出现Service的 `onDestroy` 不调用的情况。

- `startForeground` ，这个上面提到过，是通过 `Notification` 提升优先级。

- 设置 `onStartCommand()` 返回值，让服务被杀死后，系统重新创建服务，上面提到过。

五个里面就两个能稍微有点用，所以啊，网络谣传害死人。

12. IntentService

敲黑板时间，重点来了，官方强力推荐。

作，建议单独新建线程去操作，避免阻塞主线程（UI线程）

- 启动服务和停止服务是成对出现的，需要手动停止服务

IntentService完美的帮我们解决了这个问题，在内部帮我们新建的线程，不需要我们手动新建，执行完毕任务后会自动关闭。IntentService也是一个抽象类，里面有一个 `onHandleIntent(Intent intent)` 抽象方法，这个方法是在非UI线程调用的，在这里执行耗时的操作。

IntentService使用非UI线程逐一处理所有的启动需求，它在内部使用Handler，将所有的请求放入队列中，依次处理，关于Handler可以看这篇文章，也就是说IntentService不能同时处理多个请求，如果不要求服务同时处理多个请求，可以考虑使用IntentService。

IntentService在内部使用 `HandlerThread` 配合 `Handler` 来处理耗时操作。

```
private final class ServiceHandler extends Handler {
    public ServiceHandler(Looper looper) {
        super(looper);
    }

    @Override
    public void handleMessage(Message msg) {
        onHandleIntent((Intent)msg.obj);
        stopSelf(msg.arg1);
    }
}

public int onStartCommand(Intent intent, int flags, int startId) {
    onStart(intent, startId);
    return mRedelivery ? START_REDELIVER_INTENT : START_NOT_STICKY;
}

public void onStart(Intent intent, int startId) {
```

```
        mServiceHandler.sendMessage(msg);
    }
```

注意 `msg.arg1` 它是请求的唯一标识，每发送一个请求，会生成一个唯一标识，然后将请求放入 `Handler` 处理队列中，从源代码里面可以看见，在执行完毕 `onHandleIntent` 方法后，会执行 `stopSelf` 来关闭本身，同时IntentService中 `onBind()` 方法默认返回 `null` ，这说明启动**IntetService**的方式最好是用 `startService` 方法，这样在服务执行完毕后才会自动停止；如果使用 `bindService` 来启动服务，还是需要调用 `unbindService` 来解绑服务的，也需要复写 `onBind()` 方法。

> 小盆宇：在 `onHandleIntent` 中，执行 `onHandleIntent` 后紧接着执行stopSelf(int startId)，把服务就给停止了，那第一个请求执行完毕服务就停止了，后续的请求怎么会执行？

注意stopSelf(int startID)方法作用是在其参数startId跟最后启动该service时生成的id相等时才会执行停止服务，当有多个请求时候，如果发现当前请求的startId不是最后一个请求的id，那么不会停止服务，所以只有当最后一个请求执行完毕后，才回停止服务。

13. 总结

在年前写了出来，比Activity中的坑少了太多，希望对大家有帮助。下一篇是关于Android中进程通信Messenger/AIDL的，是本篇的补充，但是也属于单独的知识点。

Android 多线程：你的 **Handler** 内存泄露 了吗？

Carson_Ho ♥ 36 5

Android ：这是一份详细的 **RxJava** 实际应用案例教学

Carson_Ho ♥ 26

Android RxJava：手把手教你使用**RxJava**

Carson_Ho ♥ 10

你不会以为**Android Toast**就只是简单的吐司吧？

GreendaMi绿色大米 ♥ 48 23

插件化理解与实现 —— 加载 **Activity**「类加载篇」

梁山boy ♥ 17

评论

说说你的看法