立项:

简介

项目特别是大中型项目，要列入政府的社会和经济发展计划中。项目经过项目实施组织决策者和政府有关部门的批准，并列入项目实施组织或者政府计划的过程叫项目立项。立项分类：鼓励类、许可类、限制类，分别对应的报批程序为备案制、核准制、审批制。报批程序结束即为项目立项完成。

申请项目的[立项](https://baike.baidu.com/item/%E7%AB%8B%E9%A1%B9)时，应将立项文件递交给项目的有关审批部门。立项报告包括项目实施前所涉及的各种由文字、图纸、图片、表格、电子数据组成的材料。不同项目、不同的审批部门、不同的审批程序所要求的立项文件是各有不同的。

项目立项是国有企业或政府投资项目单位为推动某个项目上马，根据国民经济的发展、国家和地方中长期规划、产业政策、生产力布局、国内外市场、所在地的内外部条件，提出的具体项目的建议文件，是专门对拟建项目提出的框架性的总体设想，该报告的核心价值是

作用：

作为项目拟建主体上报审批部门审批决策的依据；

作为项目批复后编制项目可行性研究报告的依据；

作为项目的投资设想变为现实的投资建议的依据；

作为项目发展周期初始阶段基本情况汇总的依据。

Acticity启动模式 **launchMode:**

**standard singleTop singleTask singleInstance**

新栈模式 共享

如果应用1的任务栈中创建了MainActivity实例，如果应用2也要激活MainActivity，则不需要创建，两应用共享该Activity实例；

清栈模式

当被启动的Activity已在任务栈中会直接激活现有的activity并把activity之上的activity全部清栈

栈顶模式

当被启动的Activity已在栈顶不创建新的对象直接那现有对象使用

不在栈顶则创建新的Activity对象

普通模式

每次启动都会在栈中创建新的对象

Service 启动模式

非绑定启动 普通绑定启动 在非绑定Service之上绑定 BIND\_ABOVE\_CLINET

onCreate

↓

onBind

↓

onStartCommand

↓

onUnBind

↓

onDestroy

先绑定后启动

onCreate

↓

onStartCommand

↓

onBind

↓

onUnBind在此模式下解绑会失效

↓

onDestroy

先启动后绑定

onCreate

↓

onBind

↓

onUnBind

↓

onDeatroy

服务会跟随他所依赖的Activity生命周期

onCreate

↓

onstartCommand

↓

onDestroy

通信成本较高

一．上次的两个问题：

1.      在BindService为什么不调用onServiceDisConnection()

类ServiceConnection中的onServiceDisconnected()方法在正常情况下是不被调用的，它的调用时机是当Service服务被异外销毁时，例如内存的资源不足时这个方法才被自动调用。

Android系统在同service的连接意外丢失时调用这个．比如当service崩溃了或被强杀了．当客户端解除绑定时，这个方法不会被调用．

broadcastReceiver只能通过startService启动Service，因为广播本身生命周期很短，bind的话没有意义

2.    bindService(Intent,ServiceConnection对象，常量)

第一个参数：Intent指示对应的Service对象

第二个参数：实现了 ServiceConnection接口的对象

第三个参数：Flags

在进行服务绑定时，其标志位可以为BIND\_AUTO\_CREATE、BIND\_DEBUG\_UNBIND和BIND\_NOT\_FOREGROUND等。其中**BIND\_AUTO\_CREAT**E表示当收到绑定请求时，如果服务尚未创建，则即刻创建，在系统内存不足，需要先销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销毁对象时，服务才可被销毁；BIND\_DEBUG\_UNBIND通常用于调试场景中判断绑定的服务是否正确，但其会引起内存泄漏，因此非调试目的不建议使用；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，该标志位在Froyo中引入。

常量有:

Context. BIND\_AUTO\_CREATE:这样就会在service不存在时创建一个

automatically create the service as long

     \* as the binding exists.  Note that while this will create the service,

     \* its  {@link android.app.Service#onStartCommand}

     \* method will still only be called due toan

     \* explicit call to  {@link #startService}.  Even without that, though,

     \* this still provides you with access tothe service object while the

     \* service iscreated.

Context. BIND\_DEBUG\_UNBIND

Flag for {@link #bindService}: include debugging help for mismatched

     \* calls to unbind.  When this flag is set, thecallstackof the following

     \*  {@link #unbindService} call is retained, tobe printed if a later

     \* incorrect unbind call is made.  Note that doing this requires retaining

     \* information about the binding that wasmade for the lifetime of theapp,

     \* resulting ina leak  -- this shouldonly be used for debugging.

Context. BIND\_NOT\_FOREGROUND

Flag for {@link #bindService}: don't allow this binding to raise

     \* the target service's process to theforeground scheduling priority.

     \* It will still be raised to at least thesame memory priority

     \* as the client (so that its process willnot bekillable in any

     \* situation where the client is notkillable),but for CPU scheduling

     \* purposes it may be left in thebackground.  This only has an impact

     \* in the situation where the bindingclient is a foreground process

     \* and thetarget service is in a background process.

Context. BIND\_ABOVE\_CLIENT

Flag for {@link #bindService}: indicates that the client application

     \* binding to this service considers theservice to be more important than

     \* the app itself.  When set, the platform will try to have theout of

     \* memory kill the app before itkills the service it is bound to, though

     \* this is notguaranteed to be the case.

/\*\*

     \* Flag for  {@link #bindService}: allow the processhosting the bound

     \* service to go through its normal memorymanagement.  It will be

     \* treated more like a running service,allowing the system to

     \* (temporarily) expunge the process if lowon memory or for some other

     \* whim it may have, and being moreaggressive about making it a candidate

     \* to be killed (and restarted) if runningfor a long time.

     \*/

   publicstaticfinalintBIND\_ALLOW\_OOM\_MANAGEMENT = 0x0010;

   /\*\*

     \* Flag for  {@link #bindService}: don't impact thescheduling or

     \* memory management priority of the targetservice's hosting process.

     \* Allows the service's process to bemanaged on the background LRU list

     \* just like a regular application processin the background.

     \*/

   publicstaticfinalintBIND\_WAIVE\_PRIORITY = 0x0020;

   /\*\*

     \* Flag for  {@link #bindService}: this service is veryimportant to

     \* the client, so should be brought to theforeground process level

     \* when the client is.  Normally a process can only be raised to the

     \* visibility level by a client, even ifthat client is in the foreground.

     \*/

   publicstaticfinalintBIND\_IMPORTANT= 0x0040;

   /\*\*

     \* Flag for  {@link #bindService}: If binding from anactivity, allow the

     \* target service's process importance tobe raised based on whether the

     \* activity is visible to the user,regardless whether another flag is

     \* used to reduce the amount that theclient process's overall importance

     \* is used to impact it.

     \*/

   publicstaticfinalintBIND\_ADJUST\_WITH\_ACTIVITY = 0x0080;

   /\*\*

     \* Flag for  {@link #bindService}: Don't consider thebound service to be

     \* visible, even if the caller is visible.

     \* @hide

     \*/

publicstaticfinalintBIND\_NOT\_VISIBLE = 0x40000000;

========================================================================

二．混合使用startService和bindService方法（例子：ServiceFixDemo）

Service生命周期问题：onCreateonStartonDestroyonBind

1). StartService被启动的服务的生命周期：如果一个Service被某个Activity调用 Context.startService方法启动，那么不管是否有Activity使用bindService绑定或unbindService解除绑定到该Service，该Service都在后台运行。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。【实现启动服务，服务可以在后台长时间运行，不能和服务通信】

2). bindService被绑定的服务的生命周期：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。

【实现启动服务，服务与其启动组件有依赖关系，实现了和服务通信--Binder】

3).混合使用--被启动又被绑定的服务的生命周期：如果一个Service又被启动又被绑定，则该Service将会一直在后台运行。并且不管如何调用，onCreate始终只会调用一次，对应startService调用多少次，Service的onStart便会调用多少次。调用unbindService将不会停止Service，而必须调用stopService 或 Service的 stopSelf来停止服务

bindService和startService混合使用

a．在bind的Activity退出的时候，service会执行unBind()方法而不执行OnDestroy()方法，因为有startService方法调用过，所有Activity与Service解除绑定后会有一个与调用者没有关联的Service存在。

b．如果先bindService，再startService，再调用Context.stopService()

  Service的OnDestroy()方法不会立即执行，因为有一个与Service绑定的Activity，但是在Activity退出的时候，会执行OnDestroy,如果要立即执行stopService,就得先解除绑定。[否则应用会报错]

C.如果先执行startService，再执行bindService，结果是一样的。

【实现启动服务，服务可以在后台长时间运行，服务与其启动组件有依赖关系，实现了和服务通信--Binder】

4). 当服务被停止时清除服务：当一个Service被终止（1、调用stopService；2、调用stopSelf；3、不再有绑定的连接（没有被启动））时，onDestroy方法将会被调用，在这里你应当做一些清除工作，如停止在Service中创建并运行的线程。

特别注意：

1、你应当知道在调用bindService绑定到Service的时候，你就应当保证在某处调用 unbindService解除绑定（尽管 Activity被 finish 的时候绑定会自动解除，并且Service会自动停止）；

2、你应当注意使用 startService启动服务之后，一定要使用 stopService停止服务，不管你是否使用bindService；

3、同时使用 startService与 bindService 要注意到，Service  的终止，需要unbindService与stopService同时调用，才能终止Service，不管 startService与 bindService 的调用顺序，如果先调用 unbindService此时服务不会自动终止，再调用 stopService之后服务才会停止，如果先调用 stopService此时服务也不会终止，而再调用 unbindService或者之前调用 bindService的 Context 不存在了（如Activity 被 finish  的时候）之后服务才会自动停止；

4、Service.onBind如果返回null，则调用bindService 会启动 Service，但不会连接上 Service，因此 ServiceConnection.onServiceConnected不会被调用，但你任然需要使用unbindService函数断开它，这样Service 才会停止。

5.如果service已经启动并且接受绑定，那么当系统调用你的onUnbind()方法，你可以选择返回true表示你想在客户端下一次绑定到service时接受一个onRebind()的调用（而不是一个OnBind()的调用），OnRebind（）返回void，但是客户端依然在它的onServiceConnectionted（）回调中接收到IBinder。【**例子：**ServiceFixDemo】

6.问题：如果在一个Activity的onCreate方法中,先bindService()，再startService()，退出这个Activity时,会执行onUnBind，但是再次进入这个Activity的时候,为什么不执行onBind方法了?【**例子：**ServiceFixTwoDemo】

只有在这个Service销毁后(执行onDestory),再进这个Activity才会执行onBind。

还有就是当有两个客户端时，在第一个客户端startServie启动服务再bindService绑定服务(启动时会调用onBind()),这时跳到第二个客户端里，再客户端startServie启动服务再bindService绑定服务，启动时不会调用用onBind()了(因为之前客户端已经启动后没有onDestory()销毁Service，所以再客户端第二次绑定服务时，只会返回IBinder对象给onServiceConnected())，而且要注意的是:当第一个服务启动并绑定一个服务时，再跳去第二个服务端启动并绑定这个服务时，第二个服务端再解绑时，不会调用onUnbind()，只有回到第一个客户端时，解绑这是才会调用onUnbind(),顺序反过来结果是一样的。得出一个结论是：**当一个服务没被**onDestory()销毁之前，只有第一个启动它的客户端能调用它的onBind()和onUnbind()。

7、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），对应服务的生命周期与上述相同。

8、在 sdk 2.0及其以后的版本中，对应的 onStart已经被否决变为了 onStartCommand，不过之前的 onStart任然有效。这意味着，如果你开发的应用程序用的 sdk为 2.0 及其以后的版本，那么你应当使用 onStartCommand而不是 onStart

三．服务按运行类型分类----前台服务【创建前台服务【ServiceForeGroundDemo】

|  |  |  |
| --- | --- | --- |
| 类别 | 区别 | 应用 |
| 前台服务 | 会在通知一栏显示 ONGOING的 Notification， | 当服务被终止的时候，通知一栏的 Notification也会消失，这样对于用户有一定的通知作用。常见的如音乐播放服务。 |
| 后台服务 | 默认的服务即为后台服务，即不会在通知一栏显示 ONGOING的 Notification。 | 当服务被终止的时候，用户是看不到效果的。某些不需要运行或终止提示的服务，如天气更新，日期同步，邮件同步等。 |

后台服务我们可以自己创建 ONGOING的 Notification这样就成为前台服务吗？答案是否定的，前台服务是在做了上述工作之后需要调用 startForeground（ android2.0及其以后版本 ）或 setForeground（android 2.0以前的版本）使服务成为前台服务。这样做的好处在于，当服务被外部强制终止掉的时候，ONGOING的 Notification也被移除掉。

前台服务好处:系统在运行**后台服务**的时候，发现在手机休眠一段时间后（1-2小时），后台运行的服务被强行kill掉，有可能是系统回收内存的一种机制，要想避免这种情况可以通过startForeground让服务前台运行，当stopService的时候通过stopForeground去掉。

前台服务只是提高了服务的优先级。当然并不能保证你得 Service 永远不被杀掉，只是提高了他的优先级。【ForegroundService项目使用了反射机制来启动前台服务】

Public static intONGOING\_NOTIFICATION=1;

Notification notification=newNotification(R.drawable.icon, getText(R.string.ticker\_text),System.currentTimeMillis());

Intent notificationIntent=newIntent(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);

方法解释：

为了使服务在前台执行，需要调用 [startForeground(intnoticationID,Notication notication)](http://developer.android.com/reference/android/app/Service.html#startForeground(int, android.app.Notification)).这个参数有两个参数，一个是通知的标示，第二个是显示在状态栏中的通知。[stopForeground(boolean)](http://developer.android.com/reference/android/app/Service.html#stopForeground(boolean)).是否移除状态栏中的Notication。这个方法不能停止服务。但是，当这个服务正在运行的时候去停止服务（没有调用stopForeground()方法），这个Notication任然会被移除。【setForeground(boolean)只是简单的改变service的状态为background】

四.BindService客户端和服务端通信的几种方法总结：

.创建BindService

  如果客户端通过bindService（）方法绑定服务，此时，客户端必须提供ServiceConnection接口的实现类，该类的功能：监视客户端和服务的连接。当Android系统创建客户端与服务直接的连接，它调用ServiceConnection接口的OnServiceConnection（）方法，来发送客户端用来与服务通信的IBinder对象。

  在实现绑定服务时，最重要的方法是OnBinde（）回调方法返回的接口，有三种方法：

方法一：继承Binder类（支持跨进程）

如果**服务对应用程序私有并且与客户端运行在相同的进程中**，则应该继承Binder类来创建接口，并且从onBind()方法返回其一个实例。客户端接收到Binder对象并且用其来访问Binder类实现类或者Service类中的公共方法。【支持跨进程原因：客户端能够转型返回对象并且适当的调用其方法】

方法二：使用Messenger

Messenger:信使

官方文档解释：它引用了一个Handler对象，以便others能够向它发送消息(使用mMessenger.send(Messagemsg)方法)。该类允许跨进程间基于Message的通信(即两个进程间可以通过Message进行通信)，在服务端使用Handler创建一个Messenger，客户端持有这个Messenger就可以与服务端通信了。

以前我们使用Handler+Message的方式进行通信，都是在同一个进程中，从线程持有一个主线程的Handler对象，并向主线程发送消息。

而Android既然可以使用bindler机制进行跨进行通信，所以我们当然可以将Handler与bindler结合起来进行跨进程发送消息。

查看API就可以发现，Messenger就是这种方式的实现。

如果需要接口跨进程工作，则可以使用Messenger类来创建接口。此时，服务定义的Handler对象来响应不同类型的Message对象。Handler是Messenger的基石，能与客户端分享IBinder，允许客户端使用Message对象向服务发送命令。此外，客户端能定义自己的Message对象，这样服务能发送回消息。

使用Messenger是执行进程通信（IPC）最简单的方式。在单个线程中，Messenger类将所有的请求队列化，**这样服务每次收到一个请求**，这样就不必设计服务为线程安全。

  它引用一个Handler对象，以便others能够像它发送消息（mMessenger.send(Messagemsg)方法）。该类允许进程间基于Message的通信（即连个进程间的通信）。在服务端使用Handler创建一个Messenger，客户端持有这个Messenger就可以通信了。

Messenger实现方法：只有客户端向服务端发送消息，单向的【Demo4】

1.     远程服务端通过：Messenger mMessenger=newMessenger(mHandler);创建一个信使对象

2.     客户端通过使用bindService()请求连接连接远程

3.     远程OnBind()方法返回一个binder对象：mMessenger.getBinder();

4.     客户端使用远程返回的的binder得到一个信使（即得到远程信使）

Public voidonServiceConnection(ComponentName name,IBinder service){

    rMessenger=new Messenger(service);

}

这里new了一个Messenger对象，其实现源码：

Messenger对象  
privatefinal IMessenger mTarget;  
public Messenger(Handler target) {  
        mTarget = target.getIMessenger();  
}  
  public IBinder getBinder() {  
        return mTarget.asBinder();  
    }  
Handler对象  
  final IMessenger getIMessenger() {  
        synchronized (mQueue) {  
            if (mMessenger != null) {  
                return mMessenger;  
            }  
            mMessenger = new MessengerImpl();  
            return mMessenger;  
        }  
    }  
MessengerImpl对象  
   private final class MessengerImpl extends IMessenger.Stub {  
        public void send(Message msg) {  
            Handler.this.sendMessage(msg);  
        }  
    }

当我们调用mMessenger.getBinder()这个方法时，底层会通过mTarget对象asBinder()方法返回binder对象，而mTarget对象是我们在创建Messenger对象时通过传入其中的handler对象的getIMessenger()获取的，在Handler对象的getIMessenger()里，创建了一个MessengerImpl对象，它实现了 IMessenger.Stub由此可知Messenger也是通过aidl实现进程间通信的，mTarget=Messengerimpl，发现它的mTarget是通过aidl得到的，实际就是远程创建的那个

5.     客户端可以使用这个远程信使对象向远程发送消息：rMessenger.send(msg);

远程服务端的Handler对象就能收到消息了，然后可以调用handlerMessage(Message msg)方法中进行处理。【该Handler对象是第一步服务端创建Messenger是用的参数mHandler】

实现双向传递消息方法：

修改第5步：

//客户端的对象，服务端可以通过此对象发送消息到客户端

MessengermClientMessenger=new Messenger(new ClientHandler()); //创建客户端信使

在rMessenger.send(msg)之前通过：msg.repleyTo=mClientMessenger将自己的信使设置到消息中，这样服务端接收到消息时同时得到客户端的信使对象了，然后服务端可以在自己的Handler对象的hanlerMessage方法中接收客户端信使对象：MessengerclientMessenger=msg.replyTo得到客户端的信使对象，并向它发送消息clientMessenger.send(message);

即完成了从服务端向客户端发送消息的功能，这样客户端可以在自己的Handler对象的hanlerMessage方法中接收服务端发送的message进行处理

方法三：aidl（后续讲）

五．使用哪种方法启动服务

在什么情况下使用startService 或 bindService或同时使用startService和 bindService？

a.如果你只是想要启动一个后台服务长期进行某项任务那么使用 startService便可以了。

b.如果你想要与正在运行的 Service取得联系，那么有两种方法，一种是使用 broadcast，另外是使用bindService，前者的缺点是如果交流较为频繁，容易造成性能上的问题，并且 BroadcastReceiver本身执行代码的时间是很短的（也许执行到一半，后面的代码便不会执行），而后者则没有这些问题，因此我们肯定选择使用 bindService（这个时候你便同时在使用 startService和bindService 了，这在 Activity中更新 Service的某些运行状态是相当有用的）。

C.另外如果你的服务只是公开一个远程接口，供连接上的客服端（android的 Service 是C/S架构）远程调用执行方法。这个时候你可以不让服务一开始就运行，而只用 bindService，这样在第一次bindService 的时候才会创建服务的实例运行它，这会节约很多系统资源，特别是如果你的服务是Remote Service，那么该效果会越明显（当然在 Service创建的时候会花去一定时间，你应当注意到这点）。

**六．如何防止Android应用中的Service被系统回收?**  
对于Service被系统回收，一般做法是通过提高优先级可以解决，在AndroidManifest.xml文件中对于intent-filter可以通过android:priority= "1000"这个属性设置最高优先级，1000是最高值，如果数字越小则优先级越低，同时实用于广播，推荐大家如果你的应用很重要，可以考虑通过系统常用intent action来触发。

另外一种实现方法：上面已经讲了----前台服务