# 绑定解绑闭服务（第二种方式：ContextWrapper# bindService([Intent](eclipse-javadoc:%E2%98%82=BindService/D:%5C/Program%20Files%20%5C(x86)%5C/adt-bundle-windows-x86_64_20140101%5C/sdk%5C/platforms%5C/android-18%5C/android.jar%3Candroid.content(ContextWrapper.class%E2%98%83ContextWrapper~bindService~Landroid.content.Intent;~Landroid.content.ServiceConnection;~I%E2%98%82Intent) service, [ServiceConnection](eclipse-javadoc:%E2%98%82=BindService/D:%5C/Program%20Files%20%5C(x86)%5C/adt-bundle-windows-x86_64_20140101%5C/sdk%5C/platforms%5C/android-18%5C/android.jar%3Candroid.content(ContextWrapper.class%E2%98%83ContextWrapper~bindService~Landroid.content.Intent;~Landroid.content.ServiceConnection;~I%E2%98%82ServiceConnection) conn, int flags)）

## 绑定服务

* 创建一个类，继承android.app.Service
* 在清单文件中配置服务，跟activity的配置几乎一样。

|  |
| --- |
| <application  android:allowBackup=*"true"*  android:icon=*"@drawable/ic\_launcher"*  android:label=*"@string/app\_name"*  android:theme=*"@style/AppTheme"* >    <service android:name=*"com.example.test10.aService"*></service>  </application> |

* 使用ContextWrapper# bindService([Intent](eclipse-javadoc:%E2%98%82=BindService/D:%5C/Program%20Files%20%5C(x86)%5C/adt-bundle-windows-x86_64_20140101%5C/sdk%5C/platforms%5C/android-18%5C/android.jar%3Candroid.content(ContextWrapper.class%E2%98%83ContextWrapper~bindService~Landroid.content.Intent;~Landroid.content.ServiceConnection;~I%E2%98%82Intent) service, [ServiceConnection](eclipse-javadoc:%E2%98%82=BindService/D:%5C/Program%20Files%20%5C(x86)%5C/adt-bundle-windows-x86_64_20140101%5C/sdk%5C/platforms%5C/android-18%5C/android.jar%3Candroid.content(ContextWrapper.class%E2%98%83ContextWrapper~bindService~Landroid.content.Intent;~Landroid.content.ServiceConnection;~I%E2%98%82ServiceConnection) conn, int flags)绑定服务
* service：这是个意图对象，使用显式意图的方式即可(Intent I = new Intent(context, class))
* conn：连接对象，其实绑定服务就是与服务建立连接。需要我们继承它，并实现它的两个方法
* flags:一般传Context# BIND\_AUTO\_CREATE

## 解绑服务

* 直接调用ContextWrapper# unbindService([ServiceConnection](eclipse-javadoc:%E2%98%82=BindService/D:%5C/Program%20Files%20%5C(x86)%5C/adt-bundle-windows-x86_64_20140101%5C/sdk%5C/platforms%5C/android-18%5C/android.jar%3Candroid.content(ContextWrapper.class%E2%98%83ContextWrapper~unbindService~Landroid.content.ServiceConnection;%E2%98%82ServiceConnection) conn)，传入绑定时的conn

## 该开启服务方式的生命周期方法调用：

### 开启：

* + 如果服务没有创建，会先创建服务，然后再绑定服务
  + onCreate 🡪 onBind

### 关闭：

* 先解绑服务，在销毁服务
* onUnbind🡪 onDestroy
* 按返回键会抛error，这是正常的。其实Android也是通过抛error来解绑服务的

# 在Activity中访问服务里定义的方法

* Activity不能直接访问服务里的方法，如果要访问，需要一个中间对象，通过中间对象，访问服务里的方法
* 这个中间对象就是IBinder
* 只有bindService方式开启的服务，才能得到中间对象
* 因为，服务里的生命周期方法onBind，只有这种绑定服务的方式才会调用，

而onBind方法会返回一个IBinder对象，这个对象会传递给[ServiceConnection](eclipse-javadoc:%E2%98%82=BindService/D:%5C/Program%20Files%20%5C(x86)%5C/adt-bundle-windows-x86_64_20140101%5C/sdk%5C/platforms%5C/android-18%5C/android.jar%3Candroid.content(ContextWrapper.class%E2%98%83ContextWrapper~bindService~Landroid.content.Intent;~Landroid.content.ServiceConnection;~I%E2%98%82ServiceConnection)#onServiceConnected()方法，

这个[ServiceConnection](eclipse-javadoc:%E2%98%82=BindService/D:%5C/Program%20Files%20%5C(x86)%5C/adt-bundle-windows-x86_64_20140101%5C/sdk%5C/platforms%5C/android-18%5C/android.jar%3Candroid.content(ContextWrapper.class%E2%98%83ContextWrapper~bindService~Landroid.content.Intent;~Landroid.content.ServiceConnection;~I%E2%98%82ServiceConnection)对象，就是我们绑定服务时要给出的其中一个参数

* 我们只需要在服务里创建一个类，继承IBinder的一个子类Binder，然后定义一个方法，来调用服务里我们需要调用的方法，通过这种方式，就能间接访问服务里的方法了
* 案例：领导办证。领导(service)，平民(Activity)，秘书（中间人IBind）、
* 如果中间对象的一些方法不想被Activity调用，可以把中间对象那一部份可以被Activity访问的方法，抽取到一个接口里，而这个接口被中间对象实现。

在Activity中，使用这个接口来接收这个中间对象。这样做，Activity就只能调用接口里的方法了，而接口里的方法，就是中间对象允许被Activity访问的方法。

* 案例：还是领导办证。但是秘书（中间人IBind）除了帮平民(Activity)牵线，使其间接和领导办证外。还有一个功能，就是和领导打麻将，而打麻将这个功能是不能被平民(Activity)访问的（能访问的话，就不合适了），所以我们把可以被平民(Activity)访问的方法（办证），抽取到接口中

# 服务的混合调用（两种开启服务的方法，一起使用）

* 服务的两种开启方式是可以一起使用的，目的是让服务即可常驻后台（绑定服务是随activity死亡而死亡的，所以不能常驻后台），又可以获得中间对象，但是一定要注意顺序，不然会有bug
* 开启服务：需要先startService再bindService
* 关闭服务：需要先unbindService再stpoService

## 案例：假的音乐播放器

# 使用代码注册广播接收者

* Android四大组件都要在清单文件中注册
* 广播接收者比较特殊，既可以在清单文件中注册，也可以直接使用代码注册
* 清单文件注册：比较耗费资源
  + 广播一旦发出，系统就会去所有清单文件中寻找，哪个广播接收者的action和广播的action是匹配的，如果找到了，就把该广播接收者的进程启动起来
  + 这样的话，广播接收者在清单文件中一直存在，所以没办法关掉，除非卸载这个应用，和更改清单文件
* 使用代码注册：节省资源
  + 需要使用广播接收者时，执行注册的代码，不需要时，执行解除注册的代码
* 案例：在服务中使用代码注册广播接收者，接受屏幕锁屏和解锁广播。
  + 写在服务里，是为了能够后台运行。当服务关闭（进程结束），就注销广播接收者
  + 开启服务的时候，注册广播接收者。使用ContextWrapper#registerReceiver(receiver, intentFilter)注册广播接收者

关闭服务的时候，注销广播接收者。使用ContextWrapper#unregisterReceiver(receiver)注销广播接收者

|  |
| --- |
| **import** android.app.Service;  **import** android.content.Intent;  **import** android.content.IntentFilter;  **import** android.os.IBinder;  **public** **class** RegisterService **extends** Service {  **private** ScreenReceiver sr;  @Override  **public** IBinder onBind(Intent intent) {  **return** **null**;  }    //服务创建时注册广播接收者  @Override  **public** **void** onCreate() {  **super**.onCreate();    sr = **new** ScreenReceiver();  //创建intent-filter对象  IntentFilter filter = **new** IntentFilter();  //设置action，就是设置要接收哪个广播  filter.addAction(Intent.*ACTION\_SCREEN\_OFF*); // 屏幕关闭广播  filter.addAction(Intent.*ACTION\_SCREEN\_ON*); // 屏幕开启广播  //注册广播接收者  registerReceiver(sr, filter);  System.*out*.println("服务已开启");  }    //服务关闭时注销广播接收者  @Override  **public** **void** onDestroy() {  **super**.onDestroy();    //注销广播接收者  unregisterReceiver(sr);  System.*out*.println("服务已关闭");  }  } |

# 特殊的广播接收者

* 安卓中有一些广播接收者，必须使用代码注册，清单文件注册是无效的
* 因为这些广播发出的次数实在太多，如果广播接收者一直在清单文件中生效，就会一直接收这类广播，很耗费资源
* 所以使用代码注册，当关掉应用的进程时，就注销广播接收者

## 屏幕锁屏和解锁

* 锁屏广播的action（未知）：*ACTION\_SCREEN\_OFF*
* 解锁广播的action（未知）：*ACTION\_SCREEN\_ON*

## 电量改变

* 电量改变广播的action（未知）：*ACTION\_BATTERY\_CHANGED*

# 服务的分类

* 本地服务：指的是服务和启动服务的activity在同一个进程中
* 远程服务：指的是服务和启动服务的activity不在同一个进程中
* 启动和绑定本地服务，用的都是显式的intent。那么启动和绑定远程服务，永的就是隐式的intent

## startService远程服务

* 定义服务类，在清单文件中配置，有<intent-filter>子节点，且action也是自定义的
* 服务类定义在code\Service

## stopService远程服务

## bindService远程服务

* 定义服务类，在清单文件中配置，有<intent-filter>子节点，且action也是自定义的
* 服务类定义在code\Service

## unbindService远程服务

# AIDL

* Android interface definition language
* 安卓接口定义语言
* 用：跨进程通信
* 应用场景：远程服务中的中间人对象，其他应用是拿不到的，那么在通过绑定服务获取中间人对象时，就无法强制转换，使用aidl，就可以在其他应用中拿到中间人类所实现的接口
* 案例：办证。领导(service)，平民(Activity)，秘书（中间人IBind）。这时候的领导(service)是在另一个项目中的，就是远程服务
* 这个时候如果还用原来的方法，来访问服务中的方法时，是做不到的。因为：中间人对象使用一个接口来接收的，这个接口里有中间人对象允许给activity访问的方法。而这个接口是定义在服务项目中的，当activity拿到中间人对象，要强专程这个接口来使用的时候，是不可以的，因为这个接口不在activity的项目中
* 这个时候就要用到aidl了
* 把远程服务的方法抽取成一个单独的接口java文件（就是上面说的那个封装了中间人对象允许被activity访问的方法的接口）
* 把接口java文件的后缀名改成aidl(需要先把接口中，public关键字去掉，因为aidl默认都是public的)，这时候，系统会在gen这个目录下生成一个与接口java同名且后缀名也相同的文件
* gen这个目录里的接口java，是系统自动生成的，且在这个java文件里，有一个静态抽象类Stub，它已经继承了binder类，实现了publicBusiness接口，这个类就是新的中间人
* 我们在服务项目中，把秘书这个中间人直接继承Stub即可，然后实现接口java文件中定义的方法即可
* 把aidl文件复制粘贴到06项目，粘贴的时候注意，aidl文件所在的包名必须跟服务项目中aidl所在的包名一致
* 在activity项目中，拿到中间人对象要强转时，直接使用Stub.asInterface(IBider)，得到的是我们生成aidl文件的哪个接口java文件中定义的接口类（也就是我们要的中间人对象方法所抽取出来的接口对象）
* 这里的关键是：系统会把这两个aidl文件生成的接口java文件看成是同一个文件
* 服务项目写在了code\LeadCertificateService

# 进程优先级补充

* 前台进程
* 拥有一个正在与用户交互的activity（onResume调用）的进程
* 拥有一个与正在和用户交互的activity绑定的，服务的进程
* 拥有一个正在“运行于前台”的服务——服务的startForeground方法调用
* 拥有一个正在执行以下三个生命周期方法中任意一个的服务(onCreate(), onStart(), or onDestroy())
* 拥有一个正在执行onReceive方法的广播接收者的进程
* 可见进程
* 拥有一个不在前台，但是对用户依然可见的activity（onPause方法调用）的进程
* 拥有一个与可见（或前台）activity绑定的，服务的进程

# 样式和主题

* 样式：用在布局文件中的各种控件上的
  + 如果有多个控件的样子都一样，那么就可以把样子定义为样式。可以被多个控件所使用
  + 样式在res\values\styles.xml中定义

|  |
| --- |
| <resources>  <!--  Base application theme, dependent on API level. This theme is replaced  by AppBaseTheme from res/values-vXX/styles.xml on newer devices.  -->  <style name=*"AppBaseTheme"* parent=*"android:Theme.Light"*>  <!--  Theme customizations available in newer API levels can go in  res/values-vXX/styles.xml, while customizations related to  backward-compatibility can go here.  -->  </style>  <!-- Application theme. -->  <style name=*"AppTheme"* parent=*"AppBaseTheme"*>  <!-- All customizations that are NOT specific to a particular API-level can go here. -->  </style>    <style name=*"mystyle1"*>  <item name=*"android:layout\_width"*>wrap\_content</item>  <item name=*"android:layout\_height"*>wrap\_content</item>  <item name=*"android:textColor"*>#ff00ff</item>  <item name=*"android:textSize"*>22sp</item>  <item name=*"android:text"*>@string/hello\_world</item>  </style>    <style name=*"mystyle2"* parent=*"mystyle1"*>  <item name=*"android:textColor"*>#0000ff</item>  </style>    <style name=*"mystyle2.mystyle3"*>  <item name=*"android:textSize"*>44sp</item>  </style>  </resources> |

* 使用

|  |
| --- |
| <TextView  style=*"@style/mystyle1"* />  <TextView  style=*"@style/mystyle2"* />  <TextView  style=*"@style/mystyle2.mystyle3"* />  <TextView  style=*"@style/mystyle1"* /> |

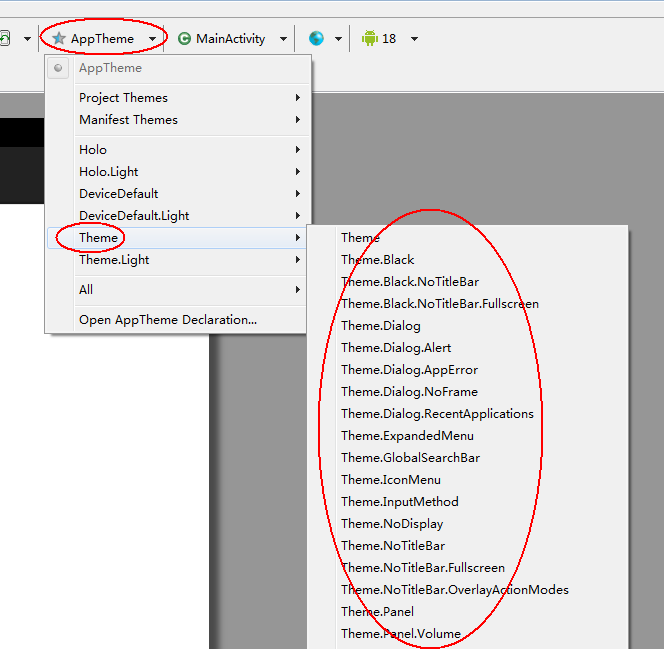
* 主题：用在清单文件的application和activity上的
* 定义方式跟样式一摸一样。有时候要分辨是样式还是主题，需要用过才知道
* 定义

|  |
| --- |
| <resources xmlns:android=*"http://schemas.android.com/apk/res/android"*>  <!--  Base application theme, dependent on API level. This theme is replaced  by AppBaseTheme from res/values-vXX/styles.xml on newer devices.  -->  <style name=*"AppBaseTheme"* parent=*"android:Theme.Light"*>  <!--  Theme customizations available in newer API levels can go in  res/values-vXX/styles.xml, while customizations related to  backward-compatibility can go here.  -->  </style>  <!-- Application theme. -->  <style name=*"AppTheme"* parent=*"AppBaseTheme"*>  <!-- All customizations that are NOT specific to a particular API-level can go here. -->  </style>    <style name=*"myt"*>  <item name=*"android:background"*>#00ff00</item>  </style>  </resources> |

* 使用

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"utf-8"*?>  <manifest xmlns:android=*"http://schemas.android.com/apk/res/android"*  package=*"com.example.service"*  android:versionCode=*"1"*  android:versionName=*"1.0"* >  <uses-sdk  android:minSdkVersion=*"10"*  android:targetSdkVersion=*"17"* />  <application  android:allowBackup=*"true"*  android:icon=*"@drawable/ic\_launcher"*  android:label=*"@string/app\_name"*  android:theme=*"@style/AppTheme"* >  <activity  android:name=*"com.example.service.MainActivity"*  android:label=*"@string/app\_name"* >  <intent-filter>  <action android:name=*"android.intent.action.MAIN"* />  <category android:name=*"android.intent.category.LAUNCHER"* />  </intent-filter>  </activity>  <service android:name=*"com.example.service.MyService"*>  <intent-filter >  <action android:name=*"my3"*/>  </intent-filter>  </service>  </application>  </manifest> |

* 在布局文件预览界面，可以切换主题



# 国际化