广播接收者
BroadcastReceiver
静态注册 在清单文件中声明对应的节点

动态注册 registerReceiver(接收者,意图过滤器) 注册之后要注销 unregisterReceiver
系统频繁发出的广播必须通过动态注册的方式

发广播
有序广播 sendOrderedBroadcast priority 声明优先级 可以修改 可以终止
无序广播 sendBroadcast

样式主题
style

国际化

对话框 在activity中创建一个对话框 上下文必须用this 当前的activity 不能用getApplicationContext
AlertDialog
AlertDialog.Builder.show();
ProgressDialog 在子线程中可以修改进度条对话框的进度

day12

1进程的概念&进程优先级

当应用运行之后 系统会创建一个linux进程 大部分情况下一个android应用对应一个linux进程
这个进程在一开始的时候只有一个线程
所有的组件都运行在同一个进程和同一个线程中(默认情况) 四大组件都运行在主线程中

Android系统会尽量保证每一个开启的进程 尽可能长的运行在手机中

进程优先级
- **1 Foreground process 前台进程**

当前进程中 有activity处于可见可操作的状态(activity执行了 onResume 之后 并且留在了这个状态 正在被用户操作)
service执行生命周期方法 以及 广播接收者(onReceive)
前台进程 几乎不会被系统杀死
- **2 Visible process 可视进程**

有activity 处于onPause()状态 可见不可操作 (透明应用盖在上面,或者是一个对话框activity盖在上面)

只有当前台进程内存不够的时候才会杀死可视进程

3 Service process 服务进程

用startservice开启了一个服务 并且运行在后台 而且没有其他组件处于前两档的状态

服务进程也很少会被杀死

4 Background process 后台进程

activity处于onstop状态 但是没有被销毁

通常会有大量的应用处于后台进程的状态 哪个应用的进程先被系统回收 系统使用LRU (least recently used) list 算法

最近最少使用 刚使用的应用最后被杀死 最早使用的那个应用最先被杀死

5 Empty process 空进程

没有任何组件运行 保存这个空进程的目的是为了缓存当前的进程,加快下次启动应用的时间

2 startservice方式开启服务 ☆☆☆☆☆

startService(Intent) 通过这种方式开启的服务 执行的声明周期方法:

第一次调用startService的时候

onCreate()->onStartCommand

再次调用startService ->onstartCommand

想停止用startservice开启的服务 stopService(intent);

stopService 执行之后 service会走 onDestroy()方法 执行之后service销毁

再次调用stopService没有反应

如果在activity中通过startService方法开启一个服务 当activity退出的时候service不会销毁 依然在后台运行

只有手动调用stopService 或者在应用管理器中关闭service 服务才会销毁

通过startservice可以提高应用的优先级

3电话录音机

新的API:

TelephonyManager 电话管理器 通过listen()方法可以获得电话的状态

PhoneStateListener 电话状态监听器 通过重写里面的方法来获取电话状态的变化

MediaRecorder 媒体录音机

获取电话的状态需要权限

```
1.  <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
```

录音需要权限

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

开机广播需要权限

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

Service代码

```
public class RecordService extends Service {

    @Override
    public IBinder onBind(Intent intent) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        //创建一个电话管理器对象 通过这个对象可以监听电话的状态
        TelephonyManager manager = (TelephonyManager) getSystemService(TELEPHONY_SERVICE);
        // PhoneStateListener 电话状态监听器
        MyPhoneStateListener listener = new MyPhoneStateListener();
        //调用电话管理器的listen方法 注册监听
        manager.listen(listener, PhoneStateListener.LISTEN_CALL_STATE);
    }

    private class MyPhoneStateListener extends PhoneStateListener {
        private MediaRecorder recorder;

        @Override
        public void onCallStateChanged(int state, String incomingNumber) {
            switch (state) {
            case TelephonyManager.CALL_STATE_IDLE:
                System.out.println("空闲状态"+"录音结束");
                if(recorder != null){
                    //停止当前录音
                    try {
                         recorder.stop();
                         //重置recorder
                         recorder.reset();   // You can reuse the object by going back to setAudioSource() step
                         recorder.release(); // Now the object cannot be reused
                    } catch (Exception e) {
                        // TODO: handle exception
                    }

```

```
                }

                break;
            case TelephonyManager.CALL_STATE_RINGING:
                System.out.println("响铃"+incomingNumber+"准备一个录音机");
                recorder = new MediaRecorder();
                //设置音频的输入源 MIC 只能录自己的声音  voice_call 录双方的声音
                 recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
                 //设置音频输出的格式 3gp
                 recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
                 //设置音频编码 amr 早期彩屏彩铃手机上使用的音频格式 一般用作手机铃声 NB narrow Band 窄带 WB wide band
                 recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
                 //设置输入之后文件保存的路径
                 recorder.setOutputFile(getCacheDir()+"/"+incomingNumber+".3gp");
                 //录音机开始准备
                 try {
                    recorder.prepare();
                } catch (IllegalStateException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                } catch (IOException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
                break;
            case TelephonyManager.CALL_STATE_OFFHOOK:
                System.out.println("接电话了"+incomingNumber+"开始录音");
                 recorder.start();   // Recording is now started
                break;

            }
        }
    }
}
```

4使用服务注册特殊广播接收者

startService 开启服务 当服务创建的时候注册广播接收者
可以把必须动态注册的广播接收者放到服务中注册

MyService.java代码

```
public class MyService extends Service {

    private BroadcastReceiver receiver;
    @Override
```

```
    public IBinder onBind(Intent intent) {
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        receiver = new ScreenReceiver();
        IntentFilter filter = new IntentFilter();
        filter.addAction(Intent.ACTION_SCREEN_ON);
        filter.addAction(Intent.ACTION_SCREEN_OFF);
        registerReceiver(receiver, filter);
    }

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

5bindservice开启服务特点 ☆☆☆☆☆

① bindService unbindService

```
Intent service = new Intent(this,BindService.class);
        conn = new MyConnection();
        //通过bind方式开启service
        //第一个参数 intent
        //第二个参数   ServiceConnection 接口 通过这个接口可以接受服务开启或者停止的消息
        //第三个参数 开启服务时候 操作的选项 一般传入BIND_AUTO_CREATE 自动创建service
        bindService(service, conn, BIND_AUTO_CREATE);
```

② bindservice之后 生命周期 onCreate->onBind 多次调用bindService onBind只会执行一次

③ activity退出的时候 必须解除跟service的绑定 在ondestroy 的时候调用 unbindService

④ unbindService多次调用会抛异常 只能调用一次

⑤ bindservice的时候传入第二个参数 是ServiceConnection 只有当 onBind方法返回不为空的时候才会调用 onServiceConnected

```
private class MyConnection implements ServiceConnection{

        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            //只有当 service的onbind方法返回值不为null 才会调用onServiceConnected
            System.out.println("onServiceConnected");

```

```
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
            //当服务正常退出的时候不会调用onServiceDisconnected
            System.out.println("onServiceDisconnected");
        }

    }
```

生命周期

bindservice 开启服务 onCreate->onbind() onBind只会执行一次

startService onCreate->onstartCommand() onstartCommand会执行多次 调用一次 startService执行一次onstartCommand

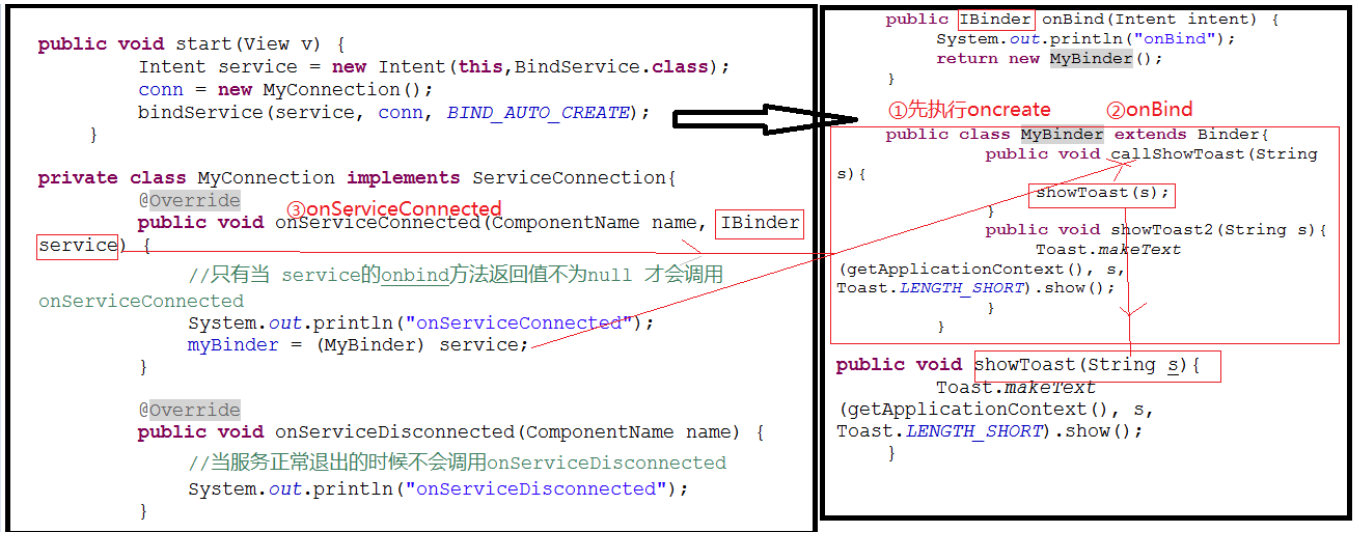
startService 开启的服务 跟activity没有关系 bindservice开启的服务 跟activity之间的关系 不求同生但求同死 activity退出的时候必须通过unbindService 关闭服务

startservice 结束的时候 stopService 可以调用多次 (只有第一次调用的时候有效)

bindservice结束 unbindService 只能调用一次 调用多次应用会抛异常

6 通过bindservice调用服务中的方法 难点

核心思路 获取Service中的内部类对象(Binder)



7 音乐播放器框架

如果把播放的逻辑放到activity 不靠谱 activity在音乐播放器中只是作为界面的作用
所以控制音乐播放的逻辑 要放到service里
Activity既能控制音乐播放的逻辑 又可以实现退出activity之后service不死掉 需要使用混合方式开启服务
混合方式开启服务 ☆☆☆☆☆
startService
bindService 一起使用 先后顺序没有要求
Activity代码

```
public class MainActivity extends Activity {

    private MyConnection conn;
    private MyBinder musicControl;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // 通过bind方式开启音乐播放的服务
        Intent service = new Intent(this, MusicPlayerService.class);
        conn = new MyConnection();
        //③调用bindService
        bindService(service, conn, BIND_AUTO_CREATE);
        //混合方式开启服务
        startService(service);
    }

    public void pre(View v) {
        musicControl.callPre();
    }

    public void play(View v) {
        musicControl.callplay();
    }

    public void pause(View v) {
        musicControl.callpause();
    }

    public void next(View v) {
        musicControl.callNext();
    }

    private class MyConnection implements ServiceConnection {

        @Override
        public void onServiceConnected(ComponentName name, IBinder service) {
            //④service执行完onBind之后 返回值不为空 就会走这个方法  IBinder对象就是 onBind的返回值  获取到这个对象 就可以调用它的public方法
            musicControl = (MyBinder) service;
            musicControl.playHiFiMusic();
```

```
            IService iservice = (IService) service;
            iservice.callpause();
            iservice.callpause();
        }

        @Override
        public void onServiceDisconnected(ComponentName name) {
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        unbindService(conn);
    }

}
```

service代码

```
public class MusicPlayerService extends Service {

    @Override
    public IBinder onBind(Intent intent) {
        //③onCreate之后执行onBind
        return new MyBinder();
    }
    public class MyBinder extends Binder implements IService{
        //播放下一首
        public void callNext(){
            next();
        }
        //播放上一首
        public void callPre() {
            pre();
        }
        //播放音乐
        public void callplay() {
            play();
        }
        //暂停
        public void callpause() {
            pause();
        }

        public void playHiFiMusic(){

        }
    }

    @Override
    public void onCreate() {
        super.onCreate();
        // MediaPlayer
```

```
        //@bindService之后执行 onCreate()
        System.out.println("准备一个音乐播放器");
    }

    public void next() {
        System.out.println("播放下一首... 小苹果");
    }

    public void pre() {
        System.out.println("播放上一首... 小苹果");
    }

    public void play() {
        System.out.println("播放上一首... 江南皮革厂 倒闭了");
    }

    public void pause() {
        System.out.println("暂停");
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        System.out.println("onDestory");
    }

}
```

8 通过接口调用服务中的方法

可以定义一个接口 暴露部分方法给调用者

IService.java 接口代码

```
public interface IService {
    //播放下一首
    //public void callNext();
    //播放上一首
    //public void callPre();
    //播放音乐
    public void callplay();
    //暂停
    public void callpause();
}
```

让Binder对象实现接口

```
public class MyBinder extends Binder implements IService{
```

在Activity的onServiceConnected方法中可以通过 service强转成 Binder 或者 强转成接口类型 实现

对暴露方法的控制(强转成接口的只能调用接口中的方法)

```
public void onServiceConnected(ComponentName name, IBinder service) {
            //④service执行完onBind之后 返回值不为空 就会走这个方法  IBinder对象就是 onBind的返回值  获取到这个对象 就可以调用它的public方法
            musicControl = (MyBinder) service;
            musicControl.playHiFiMusic();
            IService iservice = (IService) service;
            iservice.callpause();
            iservice.callpause();
        }
```

9 aidl 安卓接口定义语言

让其它应用可以调用当前应用service中的方法
RPC remote procedure call 远程过程调用 AIDL 解决就是rpc的问题
IPC inter process communication 进程间通信
每一个android应用都运行在独立的进程中 所以 应用之间的通信就是进程间通信
Activity intent
BroadCastReceiver 通过onReceive方法 可以处理其他应用发来的广播
通过Intent 携带数据

AIDL实现的过程
提供远程服务方法的应用
① 创建一个Service 重写onBind方法 在onBinder中返回一个Binder对象 需要远程调用的放发放到这个Binder对象中

```
public class RemoteService extends Service{

    @Override
    public IBinder onBind(Intent intent) {
        return new MyBinder();
    }
    public class MyBinder extends Binder{
        public void callRemoteMethod(){
            remoteMethod();
```

```
        }
    }

    public void remoteMethod(){
        System.out.println("remoteMethod is called");
    }
}
```

②在清单文件中声明对应的service 需要添加一个intent-filter 可以通过隐式意图调用service

```
<service android:name="com.itheima.remotemethod.RemoteService">
            <intent-filter >
                <action android:name="com.itheima.remoteservice"/>
            </intent-filter>
        </service>
```

③ 创建一个接口 需要暴露给其它应用调用的方法都声明在这个接口中

```
public interface IService {
public void callRemoteMethod();
}
```

④把接口文件的扩展名修改为 .aidl 需要注意 aidl文件不支持public 关键字

```
package com.itheima.remotemethod;

interface IService {
void callRemoteMethod();
}

```

如果aidl创建的没有问题 就会在gen目录下生成一个IService.java

⑤修改service的代码 让MyBinder继承Stub

```
public class MyBinder extends Stub{
        public void callRemoteMethod(){
            remoteMethod();
        }
    }
```

远程调用服务的应用

①用过隐式意图以及bindService方式 开启远程的服务

```
  Intent service = new Intent();
        //用隐式意图开启其他应用的service
        service.setAction("com.itheima.remoteservice");
        MyConnection conn = new MyConnection();
        //通过bindservice开启服务
        bindService(service, conn, BIND_AUTO_CREATE);
```

② 创建ServiceConnection的实现类

```
private class MyConnection implements ServiceConnection{
   @Override
   public void onServiceConnected(ComponentName name, IBinder service) {


   }

   @Override
   public void onServiceDisconnected(ComponentName name) {
       // TODO Auto-generated method stub

   }

  }
```

③在当前应用中创建一个目录 目录结构跟提供远程服务的应用aidl所在目录结构保持一致, 把aidl文件copy过来
如果没有问题 会在gen目录下生成一个 Iservice.java文件 包名跟aidl文件的包名一致
④在onserviceConnected方法中 通过

```
iService = Stub.asInterface(service);
```

把当前的Ibinder对象转化成远程服务中的接口类型 最终通过这个对象实现调用远程方法

```
public void callremote(View v){
      try {
       iService.callRemoteMethod();
   } catch (RemoteException e) {
       e.printStackTrace();
   }
  }
```

Service
概念
进程 ①默认情况 每一个android应用都运行在一个单独的进程中
②当应用运行起来之后 系统会创建一个进程 在这个进程中创建一个单独的线程(主线程) 四大组件都运行在相同的进程和相同的线程中
③ 进程的创建的销毁都是由系统来执行的

以及进程的优先级
① 前台进程 ②可视进程 ③ 服务进程(有startservice开启的服务在后台运行) ④后台进程(LRU 最近最少使用)⑤空进程

startService

bindservice 区别

aidl / rpc/ipc
可以让一个应用调用另外应用中服务的方法
ipc 最常用的api intent

电话录音机 startService开启服务 ☆☆☆☆☆
TelephonyManager
PhoneStateListener onCallStateChanged(int state,string incommingNumber)
MediaRecorder

音乐播放器框架 (混合的方式开启服务) ☆☆☆☆☆

混合方式开启服务 ☆☆☆☆☆
startService
bindService 一起使用 先后顺序没有要求

如果想关闭混合方式开启的服务 stopService unbindService 分别调用

aidl过程☆☆☆