

Android程序设计

服务和广播

2019.5.24

isszym sysu.edu.cn

内容

- Service
- 消息传递
- BroadcastReceiver

服务(Service)

● 概述

- Service与Activity类似，只是没有界面。使用Service要先定义一个Service子类，并在AndroidManifest.xml中配置它。
- Service和Activity一样，都是从Context派生出来的，因此，都可以使用getResources()和getContentResolver()。
- 有两种服务启动的方法：
 - (1) 直接启动Service：通过Context的startService()方法启动Service，访问者与Service之间没有关联，即使访问者退出了，Service也依然运行。
 - (2) 用绑定启动Service：通过Context的bindService()方法启动Service，访问者与Service绑定在一起，访问者退出了，Service也就终止了。
- 在绑定启动服务的方法中，有单线程和多线程两种启动方法：
在主线程中直接启动服务，如果服务时间过长会出现跳帧现象，系统会发出ANR（Application Not Responding）异常警告；用IntentService启动会把Service放在子线程中执行，可以克服了跳帧现象。

[参考](#)

● 直接启动Service

项目名: NewFirstService

MainActivity.java

```
import android.support.v7.app.AppCompatActivity; import android.widget.Button;
import android.os.Bundle; import android.content.Intent;
import android.view.View; import android.view.View.OnClickListener;
public class MainActivity extends AppCompatActivity {
    Button start, stop;
    @Override
    public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        start = (Button) findViewById(R.id.start);
        stop = (Button) findViewById(R.id.stop);
        start.setOnClickListener(new OnClickListener(){
            @Override
            public void onClick(View arg0){
                Intent intent = new Intent(this,FirstService.class);
                startService(intent);
            }
        });
        stop.setOnClickListener(new OnClickListener(){
            @Override
            public void onClick(View arg0){
                Intent intent = new Intent(this,FirstService.class);
                stopService(intent);
            }
        });
    }
}
```

FirstService.java

```
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class FirstService extends Service{
    @Override
    public IBinder onBind(Intent arg0){           // 必须实现的方法
        return null;
    }
    @Override
    public void onCreate() {                     // Service被创建时回调该方法。
        super.onCreate();
        System.out.println("Service is Created");
    }
    // Service被启动时回调该方法
    @Override
    public int onStartCommand(Intent intent, int flags, int startId){
        System.out.println("Service is Started");
        return START_STICKY;
    }
    @Override
    public void onDestroy(){                     // Service被关闭之前回调。
        super.onDestroy();
        System.out.println("Service is Destroyed");
    }
}
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal"
    >
    <Button
        android:id="@+id/start"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="start"
        />
    <Button
        android:id="@+id/stop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="stop"
        />
</LinearLayout>
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.isszym.myapplication">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service
            android:name=".FirstService"
            android:enabled="true"
            android:exported="true">
        </service>
    </application>

</manifest>
```



点击了两次START，一次STOP，run的信息：

I/System.out: Service is Created

I/System.out: Service is Started

I/System.out: Service is Started

I/System.out: Service is Destroyed

● 用绑定方式启动服务(BindService)----单线程

项目名: NewBindService

MainActivity.java

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;import android.app.Service;
import android.content.ComponentName;import android.content.Intent;
import android.content.ServiceConnection;
import android.os.IBinder;import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    Button bind, unbind, getServiceStatus;
    BindService.MyBinder binder;           // 保持所启动的Service的IBinder对象

    private ServiceConnection conn = new ServiceConnection(){
        // 与Service连接成功时回调
        @Override
        public void onServiceConnected(ComponentName name, IBinder service){
            System.out.println("--Service Connected--");
            // 获取Service的onBind方法所返回的MyBinder对象
            binder = (BindService.MyBinder) service;
        }
        // 与Service断开连接时回调该方法
        @Override
        public void onServiceDisconnected(ComponentName name) {
            System.out.println("--Service Disconnected--");
        }
    };
};
```

```

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    bind = (Button) findViewById(R.id.bind);
    unbind = (Button) findViewById(R.id.unbind);
    getServiceStatus = (Button) findViewById(R.id.getServiceStatus);
    final Intent intent = new Intent(this, BindService.class);
    bind.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View source) {
            bindService(intent, conn, Service.BIND_AUTO_CREATE);
        }
    });
    unbind.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View source) {
            unbindService(conn);
        }
    });
    getServiceStatus.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View source) {
            Toast.makeText(MainActivity.this,
                "Service的count值为: " + binder.getCount(),
                Toast.LENGTH_SHORT).show(); //②
        }
    });
}
}
}

```

BindService.java

```
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.os.Binder;

public class BindService extends Service {
    private int count;
    private boolean quit;
    private MyBinder binder = new MyBinder();    // 定义onBinder方法所返回的对象
    public BindService() { }
    @Override
    public IBinder onBind(Intent intent) {        // 绑定该Service时回调该方法
        System.out.println("Service is Binded");
        return binder;
        // TODO: Return the communication channel to the service.
        //throw new UnsupportedOperationException("Not yet implemented");
    }
    public class MyBinder extends Binder {    // 通过继承Binder来实现IBinder类
        public int getCount() {              // 获取Service的运行状态: count
            return count;
        }
    }
}
```

```

@Override
public void onCreate(){
    super.onCreate();
    System.out.println("Service is Created");
    // 启动一条线程、动态地修改count状态值
    new Thread() {
        @Override
        public void run() {
            while (!quit) {
                try {Thread.sleep(1000); }
                catch (InterruptedException e) { }
                count++;
            }
        }
    }.start();
}

@Override
public boolean onUnbind(Intent intent){ // Service被断开连接时回调该方法
    System.out.println("Service is Unbinded");
    return true;
}

@Override
public void onDestroy(){ // Service被关闭之前回调该方法。
    super.onDestroy();
    this.quit = true;
    System.out.println("Service is Destroyed");
}
}

```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal"
    >
    <Button
        android:id="@+id/bind"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="bind"
        android:textSize="14dp" />
    <Button
        android:id="@+id/unbind"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="unbind"
        android:textSize="14dp" />
    <Button
        android:id="@+id/getServiceStatus"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="getServiceStatus"
        android:textSize="14dp" />
</LinearLayout>
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.isszym.newbindservice">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service
            android:name=".BindService"
            android:enabled="true"
            android:exported="true"></service>
    </application>

</manifest>
```



多次点击BIND，只有第一次有输出，然后点击GETSERVICESTATUS，得到左图，最后点击一次UNBIND，run的信息：

03/20 21:52:11: Launching app
I/System.out: Service is Created
I/System.out: Service is Bound
I/System.out: --Service Connected--
I/System.out: Service is Unbinded
I/System.out: Service is Destroyed

} 点击
BIND

} 点击
UNBIND

● 启动意图服务(IntentService)---多线程

一般的服务是在主线程里启动，如果服务执行时间太长，系统会发出跳帧警告，采用IntentService可以把服务在子线程中启动。

项目：NewIntentService

MainActivity.java

```
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;import android.os.Bundle;
import android.view.View;import android.app.Activity;
import android.content.Intent;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    public void startService(View source) {
        Intent intent = new Intent(this, MyService.class);
        startService(intent); // 启动Service
    }

    public void startIntentService(View source){
        Intent intent = new Intent(this, MyIntentService.class);
        startService(intent); // 启动IntentService
    }
}
```


MyIntentService.java

```
import android.app.IntentService;
import android.content.Intent;
import android.content.Context;

public class MyIntentService extends IntentService {
    public MyIntentService() {
        super("MyIntentService");
    }

    // IntentService会使用单独的线程来执行该方法的代码
    @Override
    protected void onHandleIntent(Intent intent) {
        // 该方法内可以执行任何耗时任务，比如下载文件等，此处只是让线程暂停20秒
        long endTime = System.currentTimeMillis() + 20 * 1000;
        System.out.println("---IntentService onStart---");
        while (System.currentTimeMillis() < endTime) {
            synchronized (this) {
                try {
                    wait(endTime - System.currentTimeMillis());
                }
                catch (Exception e) {
                }
            }
        }
        System.out.println("---IntentService完成任务---");
    }
}
```

MyService.java

```
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

public class MyService extends Service {
    public MyService() {
    }
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        long endTime = System.currentTimeMillis() + 20 * 1000;
        System.out.println("---Service onStart---");
        while (System.currentTimeMillis() < endTime) {
            synchronized (this) {
                try {
                    wait(endTime - System.currentTimeMillis());
                }
                catch (Exception e) {
                }
            }
        }
        System.out.println("---Service完成任务---");
        return START_STICKY; //被系统终止后自动重启服务
    }
}
```

activity_main.xml

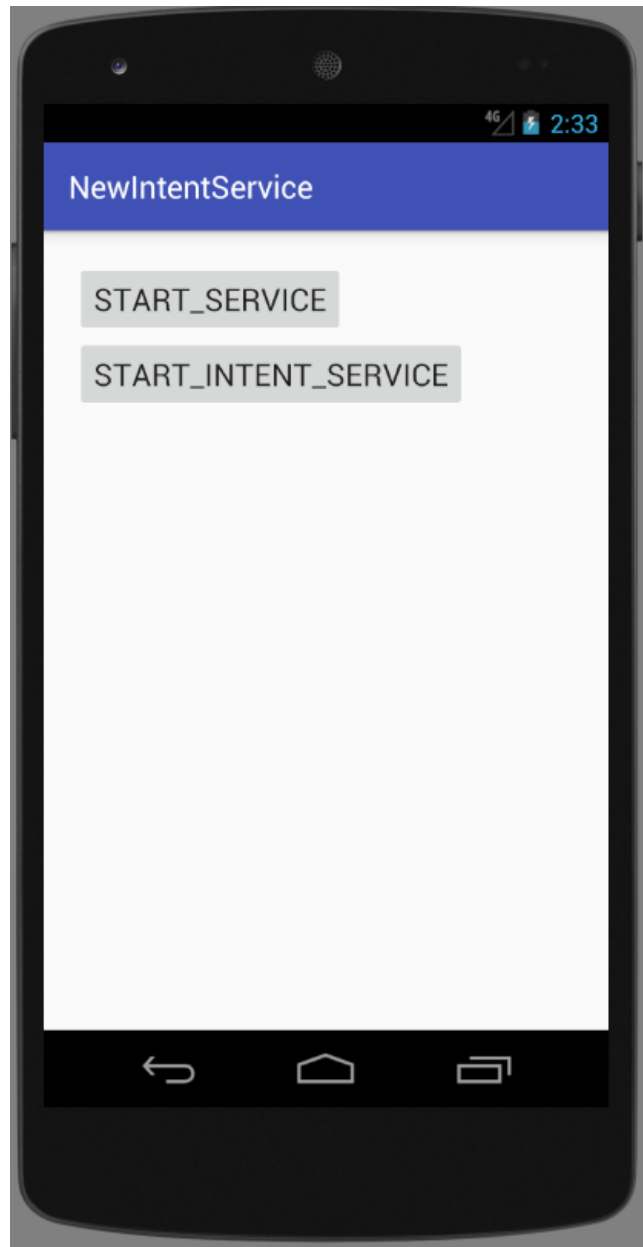
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="20dp"
    android:orientation="vertical">
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="startService"
        android:textSize="20sp"
        android:text="start_service"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="startIntentService"
        android:text="start_intent_service"
        android:textSize="20sp" />
</LinearLayout>
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.isszym.newintentservice">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <service android:name=".MyIntentService" />
            <service android:name=".MyService" />
        </activity>
        <service
            android:name=".MyIntentService"
            android:exported="false" />
        <service
            android:name=".MyService"
            android:enabled="true"
            android:exported="true"></service>
    </application>
</manifest>
```

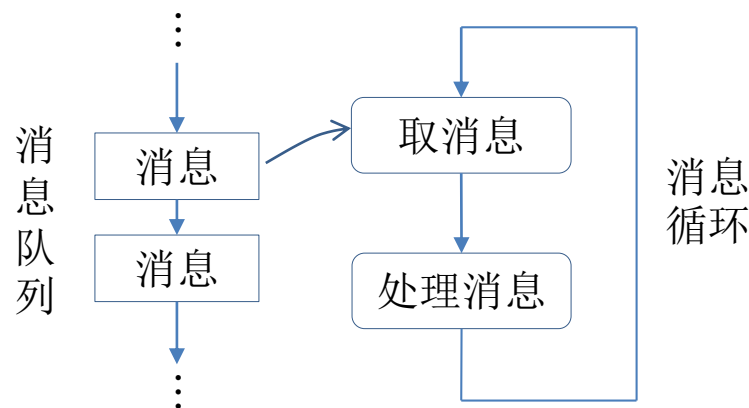


每次从*onStart*到*任务完成*都是20秒，run的信息：

- I/System.out: ---IntentService onStart---
 - I/System.out: ---IntentService完成任务---
 - I/System.out: ---IntentService onStart---
 - I/System.out: ---IntentService完成任务---
 - I/System.out: ---IntentService onStart---
 - I/System.out: ---IntentService完成任务---
 - I/System.out: ---Service onStart---
 - I/System.out: ---Service完成任务---
 - I/Choreographer: **Skipped 1200 frames! The application may be doing too much work on its main thread.**
 - I/System.out: ---Service onStart---
 - I/System.out: ---Service完成任务---
 - I/Choreographer: **Skipped 1199 frames! The application may be doing too much work on its main thread.**
 - I/System.out: ---IntentService onStart---
 - I/System.out: ---IntentService完成任务---
- 连续点击了三次
Start_Intent_Service
- 点击了一次Start_Service
- 点击了一次Start_Service，马上
又点击了Start_Intent_Service

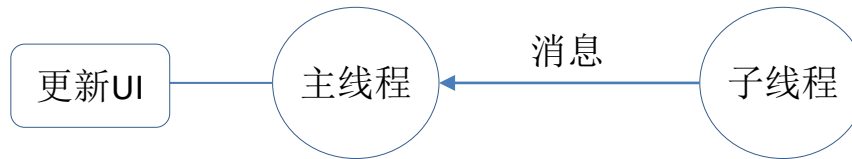
消息传递

- 安卓系统只能在主线程（UI线程）更新视图（View），而不能在子线程中直接更新视图。子线程要通过发送消息（Message）给主线程，让主线程更新视图。
- 消息传递是线程通信的一种方法。进程通信不能使用这种方法。
- 安卓系统采用消息队列（Message Queue）和消息循环（Message Loop）实现的消息处理机制。
- 主线程启动时系统会自动为其创建一个消息队列和消息循环，而子线程默认是没有的消息队列和消息循环。
- **Looper对象**负责管理线程的消息队列和消息循环。子线程通过**Looper.myLooper()**得到Looper对象，而主线程要通过**Looper.getMainLooper()**得到。子线程要执行**Looper.prepare()**创建消息队列，然后调用**Looper.loop()**来启动消息循环。
- 线程要通过Handler对象来访问Looper对象并发送和接收消息。
- **Looper对象对队列的操作是受同步保护的。只有在有消息循环的线程中才可以创建Handler对象。**



- 主线程与子线程有多种消息传递方式：

- (1) 主线程创建Handler对象来处理消息，子线程通过Handler对象向主线程发送消息，让主线程更新UI。用这种方式可以把繁重的计算任务交给子线程去完成。



- (2) 主线程向子线程传递消息。由于子线程默认没有消息循环，子线程需要创建和执行消息循环。



- (3) 创建Handler对象时可以指定Looper对象，Handler对象通过该Looper对象传递消息。
- (4) 可以在主线程的各个组件之间传递消息。
- (5) 把Runnable对象加入消息循环，由该消息循环启动该线程。

- 消息格式

```
public final class Message implements Parcelable {  
    public int what;           // 用户自定义消息编码  
    public int arg1;           // 参数1--data的补充(开销低)  
    public int arg2;           // 参数2--data的补充(开销低)  
    public Object obj;         // 对象---data的补充(开销低)  
    Bundle data;  
    int flags;  
    long when;                 // 延迟时间  
    Handler target;  
    Message next;  
    public void sendToTarget() {  
        target.sendMessage(this);  
    }  
}
```

- 创建Handler实例的时候会取到Looper类的消息循环和消息队列

```
public Handler(Callback callback, boolean async) {  
    ...  
    mLooper = Looper.myLooper();  
    mQueue = mLooper.mQueue;  
    ...  
}
```

- Handler类的把消息加入消息队列

```
boolean enqueueMessage(MessageQueue queue, Message msg, long uptimeMillis)
```

- Handler类的发送消息时用enqueueMessage加入消息:

```
boolean sendEmptyMessage(int what)
```

```
boolean sendMessage(Message msg)
```

```
boolean sendMessageDelayed(Message msg, long delayMillis)
```

```
boolean sendEmptyMessageDelayed(int what, long delayMillis)
```

```
boolean sendMessageAtTime(Message msg, long uptimeMillis)
```

- Handler类的生成消息的方法:

```
Message obtainMessage(int what, int arg1, int arg2, Object obj)
```

```
Message obtainMessage(int what, Object obj)
```

• 项目 NewSendMsgToMainThrd

子线程通过消息数据或共享变量向主线程发送消息。

```
public class MainActivity extends AppCompatActivity {
    private static final int COMPLETED = 0x123;
    private static final int DONE = 0x124;
    private int cnt = 0;
    private TextView stateText;
    private Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            switch(msg.what) {
                case COMPLETED:
                    stateText.setText("Finished! " + cnt);
                    break;
                case DONE:
                    String body = (String) msg.obj;
                    stateText.setText(body);
                    break;
            }
        }
    };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        stateText = (TextView) findViewById(R.id.tv);
    }
}
```

- (1) 消息循环会从消息队列中取出消息msg。
- (2) 调用该消息的handler (msg.target) 的 dispatchMessage进行处理。
- (3) dispatchMessage会调用 handleMessage进行处理。

```

public void onClick1(View v) {
    new Thread() {
        @Override
        public void run() {
            try { Thread.sleep(2000); } catch (Exception e) {
            }; //耗时的操作
            synchronized(this) {
                cnt++; //利用共享变量在主线程和子线程之间传递数据
            }
            handler.sendMessage(COMPLETED);
        }
    }.start();
}

public void onClick2(View v) {
    new Thread() {
        @Override
        public void run() {
            try { Thread.sleep(2000); } catch (Exception e) {}; //耗时的操作
            synchronized(this) {
                cnt++; //利用共享变量在主线程和子线程之间传递数据
            }
            handler.obtainMessage(DONE, "Hello! " + cnt).sendToTarget();
        }
    }.start();
}
}

```



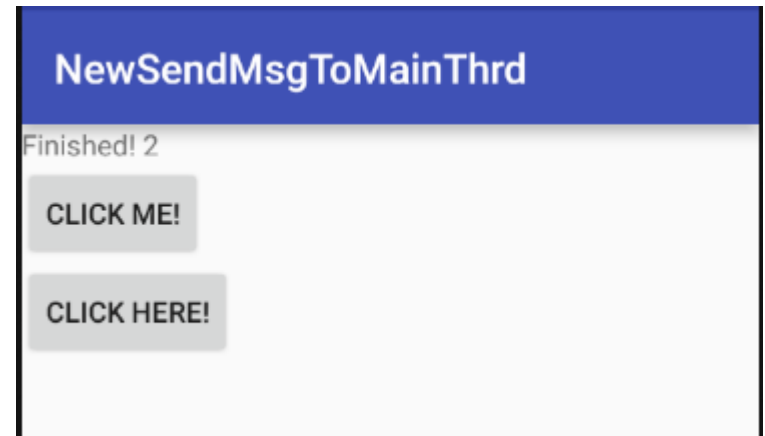
用bundle传数据见下一个例子

```

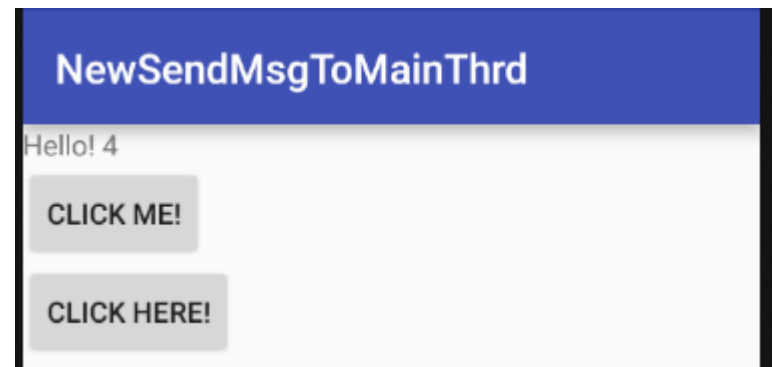
activity_main.xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/tv"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
    <Button
        android:id="@+id/btn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click me!"
        android:onClick="onClick1"/>
    <Button
        android:id="@+id/btn2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click here!"
        android:onClick="onClick2"/>
</LinearLayout>

```

连续点击两次CLICK ME之后（要等4秒）



点击两次CLICK HERE之后（要等4秒）



- 工程名 **NewSendMsgToSubthrd** 主线程通过子线程的Handler发送消息给子线程。

```
public class MainActivity extends AppCompatActivity {
    private static final int SEND = 0x100;
    private static final int COMPLETED = 0x123;
    private TextView stateText;
    private Button btn;
    int count = 0;
    WorkThread thr = new WorkThread(); // 建立新线程

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        stateText = (TextView) findViewById(R.id.tv);
        thr.start(); // 启动线程
    }

    public void click1(View v) {
        if (thr == null) return;
        count++;
        stateText.setText("count: " + count);
        Message msg = new Message();
        Bundle bundle = new Bundle();
        bundle.putString("status", "Send " + count + "!");
        msg.setData(bundle); // 消息可以带数据
        msg.what = SEND; // 消息自定义类型. 用arg1, arg2可以定义子类型
        thr.mHandler.sendMessage(msg); // 使用子线程Handler发送消息
    }
}
```

主线程通过消息的Data属性
(Bundle)向子线程传递数据

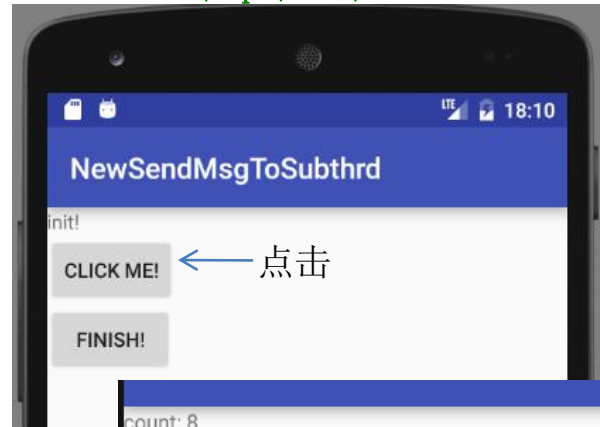
```

public void click2(View v) {
    if (thr == null) return;
    stateText.setText("Finished!");
    Message msg = new Message();
    msg.what = COMPLETED;
    thr mHandler.sendMessage(msg);
    thr = null;
}
class WorkThread extends Thread {
    public Handler mHandler;
    public void run() {
        Looper.prepare(); //创建消息队列
        mHandler = new Handler() {
            public void handleMessage(Message msg) { //处理消息队列中收到的消息
                if (msg.what == SEND) {
                    Log.i("test", msg.getData().getString("status"));
                } else {
                    if (msg.what == COMPLETED) {
                        Looper.myLooper().quit(); // 结束消息循环。
                    }
                    //版本大于18时可以用safetyQuit()
                }
            }
        };
        Looper.loop(); //启动消息循环
    }
}
}

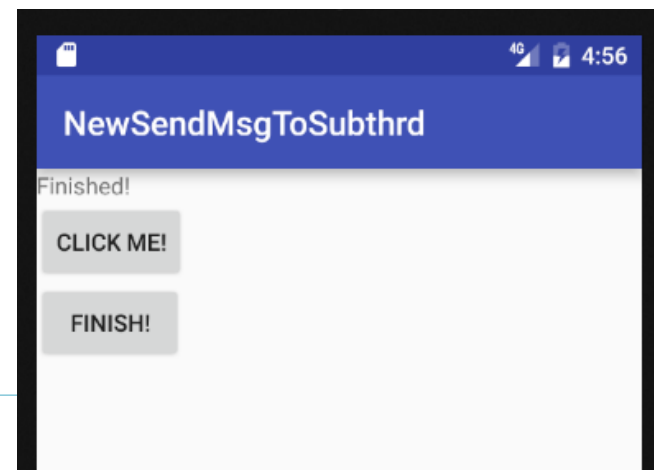
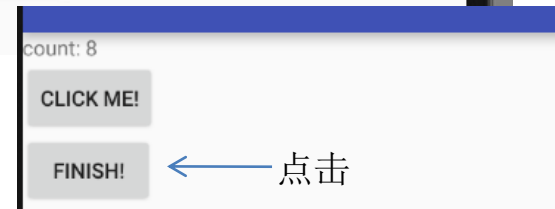
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/tv"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="init!" />
    <Button
        android:id="@+id/btn1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click Me!"
        android:onClick="click1"/>
    <Button
        android:id="@+id/btn2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Finish!"
        android:onClick="click2"/>
</LinearLayout>
```



I/test: Send 1!
I/test: Send 2!
I/test: Send 3!
I/test: Send 4!
I/test: Send 5!
I/test: Send 6!
I/test: Send 7!
I/test: Send 8!



- 工程名 **NewHandlerMainLooper**: 选择子线程或主线程Looper创建Handler。

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    TextView tv1 = null;
    MyThread thrd;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tv1 = (TextView) findViewById(R.id.tv1);
        thrd = new MyThread();
        thrd.start();
    }
    private class MyHandler extends Handler {
        public MyHandler(Looper looper) {
            super(looper);
        }
        @Override
        public void handleMessage(Message msg) { // 处理消息
            if (msg.what == 0x100)
                tv1.setText("Hi!  -- From Main Looper");
            else if (msg.what == 0x123)
                tv1.setText("Hi!  -- From Sub Looper");
        }
    }
}
```

```

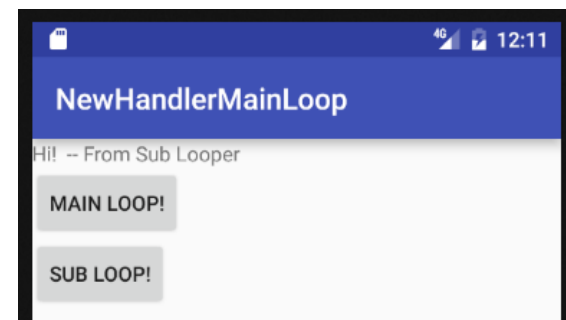
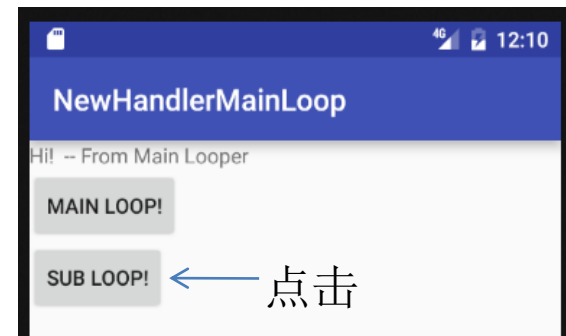
public void click(View v) {
    switch (v.getId()) {
        case R.id.btn1:
            new MyHandler(Looper.getMainLooper()).sendMessage(0x100);
            break;
        case R.id.btn2:
            new MyHandler(thrd.getLooper()).sendMessage(0x123);
    }
}

private class MyThread extends Thread {
    public Looper getLooper() {
        return Looper.myLooper();
    }
    @Override
    public void run() {
        Looper.prepare();
        Looper.loop();
    }
}
}

```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/tv1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
    <Button
        android:id="@+id/btn1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Main Loop!"
        android:onClick="click" />
    <Button
        android:id="@+id/btn2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Sub Loop!"
        android:onClick="click" />
</LinearLayout>
```



- 工程名 **NewHandlerAllInMain**: 发送消息和处理消息都在主线程进行

MainActivity.java

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {
    Button button;
    TextView text;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        button = (Button) findViewById(R.id.btn1);
        button.setOnClickListener(MainActivity.this);
        text = (TextView) findViewById(R.id.tv1);
    }
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.btn1:
                Looper loop = Looper.myLooper(); // 取得当前线程里的Looper
                MyHandler mHandler = new MyHandler(loop);
                // 构造一个handler 使之可与Looper 通信
                // button 等组件可以由mHandler 将消息传给Looper 后,
                // 再放入messageQueue 中, 同时mHandler 也可以接受来自Looper 消息
                mHandler.removeMessages(0); // 清除what=0的消息
                String msgStr = " 主线程不同组件通信: 消息来自button";
                Message m = mHandler.obtainMessage(1, 1, 1, msgStr); // 构造消息
                mHandler.sendMessage(m); // 发送消息
                break;
        }
    }
}
```

`Message obtainMessage(int what, int arg1, int arg2, Object obj)`

```
private class MyHandler extends Handler {  
    public MyHandler(Looper looper) {  
        super(looper);  
    }  
  
    @Override  
    public void handleMessage(Message msg) { // 处理消息  
        text.setText(msg.obj.toString());  
    }  
}  
}
```

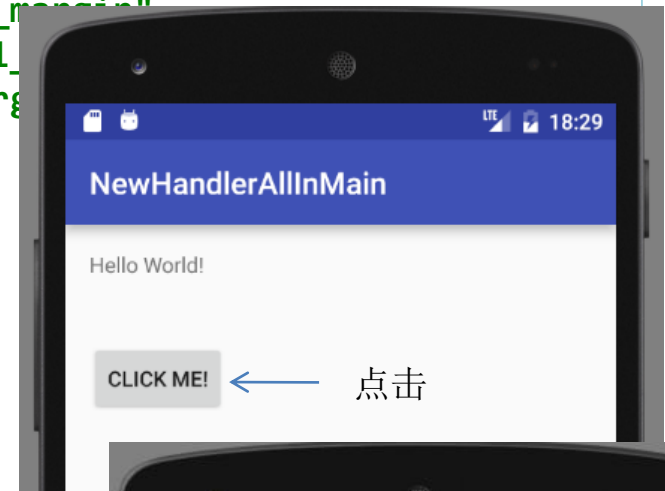
```

activity_main.xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"

    <TextView
        android:id="@+id/tv1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />

    <Button
        android:id="@+id/btn1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click Me!"
        android:layout_below="@+id/tv1"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="41dp" />
</RelativeLayout>

```



- 工程名 **NewHandlerRunnable**: 在主线程的消息队列中启动Runnable子类的实例。

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    TextView mText; int count= 0;
    Handler mHandler = new Handler(){
        @Override public void handleMessage(Message msg) { // 处理消息
            if (msg.what == 0x123)
                mText.setText("Hi!  -- From Sub Looper");
        }
    };
    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mText = (TextView) findViewById(R.id.text0); // 一个TextView
    }
    public void click1(View vw) {
        mHandler.postDelayed(r, 3000); // 3秒后把r加入循环队列
    };
    public void click2(View vw) {
        mHandler.removeCallbacks(r); // 停止线程
    }
    Runnable r = new Runnable() {
        @Override public void run() {
            Message msg = mHandler.obtainMessage();
            msg.what = 0x123;
            mHandler.sendMessage(msg);
        }
    };
}
```

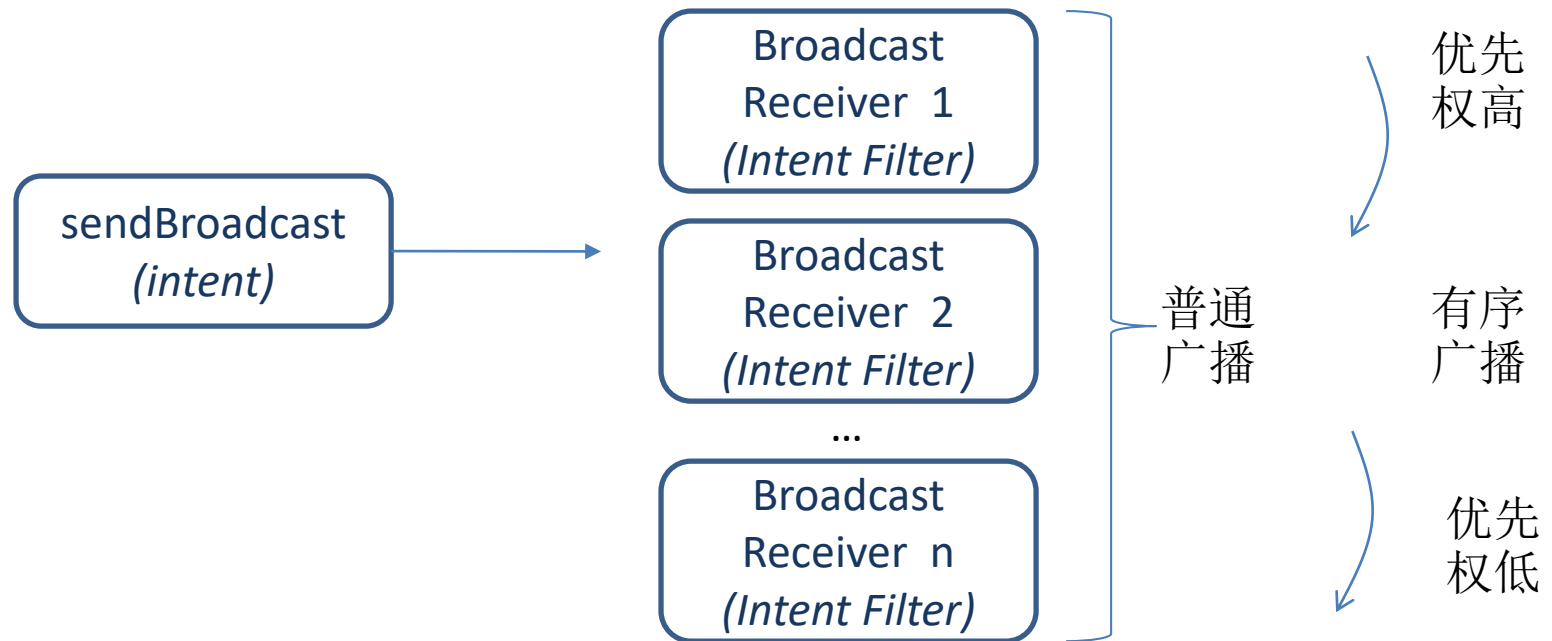
activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <TextView
        android:id="@+id/text0"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!" />
    <Button
        android:id="@+id/btn1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="add thread!"
        android:onClick="click1" />
    <Button
        android:id="@+id/btn2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="stop thread!"
        android:onClick="click2" />
</LinearLayout>
```



BroadcastReceiver

用带有Intent匹配条件的组件用sendBroadcast发出广播消息，所有匹配了Intent条件的接收者都可以收到广播消息。



普通广播是所有匹配的接收者均可同时收到广播消息，而有序广播让广播消息根据接收者的优先权依次经过每个接收者，每个接收者都可以停止继续传播消息，高优先级的Receiver通过setResultExtras方法来发送数据，低优先级的接受者通过getResultExtras来获取Bundle数据包，进而获取数据，他们都可以用intent的Bundle来获取最初的广播发送者发出的消息。

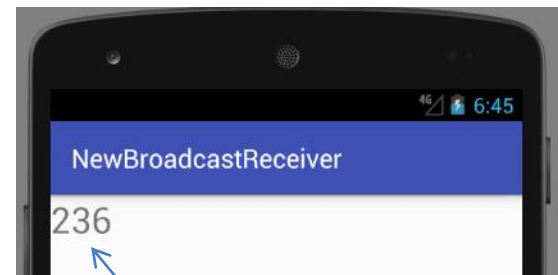
[参考](#)

[参考](#)

项目名: NewBroadcastReceiver

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    private TextView tv1 = null;
    private MyReceiver receiver = null;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tv1 = (TextView) findViewById(R.id.tv1);
        receiver = new MyReceiver();
        IntentFilter filter = new IntentFilter();
        filter.addAction("com.example.activity.CountService");
        MainActivity.this.registerReceiver(receiver, filter);
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        unregisterReceiver(receiver);
    }
    public class MyReceiver extends BroadcastReceiver {
        @Override
        public void onReceive(Context context, Intent intent) {
            Bundle bundle = intent.getExtras();
            int count = bundle.getInt("count");
            tv1.setText(""+count);
        }
    }
}
```



计数值

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/tv1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="" />
</LinearLayout>
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.isszym.newbroadcastreceiver">
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

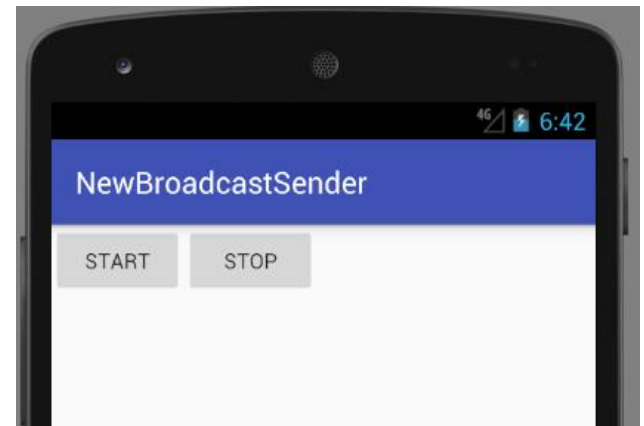
项目名称: NewBroadcastSender

MainActivity.java

```
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState); setContentView(R.layout.activity_main);  
    }  
    protected void startService(View v) {  
        Intent intent = new Intent(this, CountService.class); startService(intent);  
    }  
    protected void stopService(View v) {  
        Intent intent = new Intent(this, CountService.class); stopService(intent);  
    }  
}
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:orientation="horizontal" >  
    <Button  
        android:text="Start"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:onClick="startService" />  
    <Button  
        android:text="Stop"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:onClick="stopService" />  
</LinearLayout>
```



CountService.java

```
public class CountService extends Service {
    private int count = 0;
    private boolean threadDisable=false;
    @Override
    public IBinder onBind(Intent intent) {
        throw new UnsupportedOperationException("Not yet implemented");
    }
    @Override
    public void onCreate() {
        super.onCreate();
        threadDisable=false;
        System.out.println("Service is Created");
        new Thread(new Runnable() {
            @Override
            public void run() {
                while (!threadDisable) {
                    try {
                        Thread.sleep(1000);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    count++;
                    System.out.println(""+count);
                    Intent intent = new Intent();
                    intent.putExtra("count", count);
                    intent.setAction("com.example.activity.CountService");
                    sendBroadcast(intent);    //发送广播
                }
            }
        }).start();
    }
}
```

```

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    System.out.println("Service is Started");
    return START_STICKY; //被系统终止后自动重启
}
@Override
public void onDestroy() { // Service被关闭之前回调。
    threadDisable=true;
    super.onDestroy();
}
}

```

AndroidManifest.xml

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.isszym.newbroadcastsender">
    <application
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service
            android:name=".CountService"
            android:enabled="true"
            android:exported="true"></service>
    </application>
</manifest>

```

项目名: NewSortedBroadcast

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void onClick(View v) {
        Intent intent = new Intent();
        intent.setAction("com.example.action.SORTED_BROADCAST");
        intent.putExtra("main", "呵呵, all Receivers, 你们好!");
        sendOrderedBroadcast(intent, null); // 第二个参数为String receiverPermission,
        // 要求接收器必须具有相应权限才能正常接收到广播, 取值null时没有权限。
    }
}
```

MyReceiver.java

```
public class MyReceiver extends BroadcastReceiver{
    @Override
    public void onReceive(Context context, Intent intent){
        Toast.makeText(context, intent.getStringExtra("main"), Toast.LENGTH_LONG).show();
        Bundle bundle = new Bundle();
        bundle.putString("first", "哈哈, Receiver 2, 你好!");
        setResultExtras(bundle);
        // abortBroadcast(); /* 取消Broadcast的继续传播 */
    }
}
```

MyReceiver2.java

```
public class MyReceiver2 extends BroadcastReceiver{
    @Override
    public void onReceive(Context context, Intent intent) {    // intent为最初的intent
        // getResultExtras的参数: 为true, 没数据则创建空Bundle, 为false, 没数据则返回null
        Bundle bundle = getResultExtras(true);
        String first = bundle.getString("first"); // 解析前一个BroadcastReceiver所存入的消息
        Toast.makeText(context, first, Toast.LENGTH_LONG).show();
        Toast.makeText(context, intent.getStringExtra("main"), Toast.LENGTH_LONG).show();
    }
}
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    >
    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Send"
        android:onClick="onClick"
        />
</LinearLayout>
```



点击SEND后显示了三条信息

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.isszym.newsortedbroadcast">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <receiver android:name=".MyReceiver">
            <intent-filter android:priority="20"> <!-- -1000~1000 -->
                <action android:name="com.example.action.SORTED_BROADCAST" />
            </intent-filter>
        </receiver>
        <receiver android:name=".MyReceiver2">
            <intent-filter android:priority="0">
                <action android:name="com.example.action.SORTED_BROADCAST" />
            </intent-filter>
        </receiver>
    </application>

</manifest>
```



常见的系统广播

[参考](#)

- 系统启动完成

如果需要在开机启动完成之后启动消息推送服务功能，就要订阅系统“启动完成”这条广播。

```
public class BootCompleteReceiver extends BroadcastReceiver {  
    private static final String TAG = "BootCompleteReceiver";  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        Intent service = new Intent(context, MsgPushService.class);  
        context.startService(service);  
        Log.i(TAG, "Boot Complete. Starting MsgPushService...");  
    }  
}
```

```
<receiver android:name=".BootCompleteReceiver">  
    <intent-filter>  
        <!-- 注册开机广播地址-->  
        <action android:name="android.intent.action.BOOT_COMPLETED"/>  
        <category android:name="android.intent.category.DEFAULT" />  
    </intent-filter>  
</receiver>
```

- 网络状态变化

当网络突然断开，通过接收系统广播消息对用户发出提醒并采取相应操作。

```
public class NetworkStateReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {    //network state changed
        if (!isNetworkAvailable(context))
            Toast.makeText(context, "network disconnected!", 0).show();
    }
    public static boolean isNetworkAvailable(Context context) {
        ConnectivityManager mgr = (ConnectivityManager)
            context.getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo[] info = mgr.getAllNetworkInfo();
        if (info != null) {
            for (int i = 0; i < info.length; i++) {
                if (info[i].getState() == NetworkInfo.State.CONNECTED) return true;
            }
        }
        return false;
    }
}

<receiver android:name=".NetworkStateReceiver">
    <intent-filter>
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</receiver>
```

- 电量变化

```
public class BatteryChangedReceiver extends BroadcastReceiver {  
    private static final String TAG = "BatteryChangedReceiver";  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        // 当前电量  
        int currLevel = intent.getIntExtra(BatteryManager.EXTRA_LEVEL, 0);  
        // 总电量  
        int total = intent.getIntExtra(BatteryManager.EXTRA_SCALE, 1);  
        int percent = currLevel * 100 / total;  
        Log.i(TAG, "battery: " + percent + "%");  
    }  
}
```

```
<receiver android:name=".BatteryChangedReceiver">  
    <intent-filter>  
        <action android:name="android.intent.action.BATTERY_CHANGED" />  
        <category android:name="android.intent.category.DEFAULT" />  
    </intent-filter>  
</receiver>
```

- 监听SD卡状态

```
public class SDCardReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction(); // 区分接收到的是哪个广播
        if(action.equals("android.intent.action.MEDIA_MOUNTED")) {
            System.out.println("sd卡就绪");
        }
        else if(action.equals("android.intent.action.MEDIA_UNMOUNTED")) {
            System.out.println("sd卡被移除");
        }
        else if(action.equals("android.intent.action.MEDIA_REMOVED")) {
            System.out.println("sd卡被拔出");
        }
    }
}
```

```
<receiver android:name="com.leoyanblog.sdcardlistener.SDCardReceiver">
    <intent-filter >
        <action android:name="android.intent.action.MEDIA_MOUNTED"/>
        <action android:name="android.intent.action.MEDIA_UNMOUNTED"/>
        <action android:name="android.intent.action.MEDIA_REMOVED"/>
        <data android:scheme="file"/>
    </intent-filter>
</receiver>
```

- 监听应用安装变化

```
public class AppReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        //区分接收到的是哪种广播
        String action = intent.getAction();
        //获取广播中包含的应用包名
        Uri uri = intent.getData();
        if(action.equals("android.intent.action.PACKAGE_ADDED")) {
            System.out.println(uri + "被安装了");
        }
        else if(action.equals("android.intent.action.PACKAGE_REPLACED")) {
            System.out.println(uri + "被更新了");
        }
        else if(action.equals("android.intent.action.PACKAGE_REMOVED")) {
            System.out.println(uri + "被卸载了");
        }
    }
}
```

```
<receiver android:name="com.leoyanblog.app.AppReceiver">
    <intent-filter >
        <action android:name="android.intent.action.PACKAGE_ADDED"/>
        <action android:name="android.intent.action.PACKAGE_REPLACED"/>
        <action android:name="android.intent.action.PACKAGE_REMOVED"/>
        <data android:scheme="package"/>
    </intent-filter>
</receiver>
```

- 常见的 Action 常量

ACTION_TIME_CHANGED	系统时间被改变
ACTION_DATE_CHANGED	系统日期被改变
ACTION_TIMEZONE_CHANGED	系统时区被改变
ACTION_BOOT_COMPLETED	系统启动完成
ACTION_PACKAGE_ADDED	系统添加包
ACTION_PACKAGE_CHANGED	系统的包改变
ACTION_PACKAGE_REMOVED	系统的包被删除
ACTION_PACKAGE_RESTARTED	系统的包被重启
ACTION_PACKAGE_DATA_CLEARED	系统的包数据被清空
ACTION_BATTERY_CHANGED	电池电量改变
ACTION_BATTERY_LOW	电池电量低
ACTION_POWER_CONNECTED	系统连接电源
ACTION_POWER_DISCONNECTED	系统与电源断开
ACTION_SHUTDOWN: 系统被关闭。	

附录1、系统的Looper类

系统的Looper类和消息循环Looper.loop()

```
public class Looper {
    // 每个线程中的Looper对象其实是一个ThreadLocal，即线程本地存储(TLS)对象
    private static final ThreadLocal sThreadLocal = new ThreadLocal();
    // Looper内的消息队列
    final MessageQueue mQueue;
    // 当前线程
    Thread mThread;
    // ... 其他属性

    // 每个Looper对象中有它的消息队列，和它所属的线程
    private Looper() {
        mQueue = new MessageQueue();
        mRun = true;
        mThread = Thread.currentThread();
    }
    // 我们调用该方法会在调用线程的TLS中创建Looper对象
    public static final void prepare() {
        if (sThreadLocal.get() != null) {
            // 试图在有Looper的线程中再次创建Looper将抛出异常
            throw new RuntimeException("Only one Looper may be created per thread");
        }
        sThreadLocal.set(new Looper());
    }
}
```



```

public static final void loop() {
    Looper me = myLooper(); //得到当前线程Looper
    MessageQueue queue = me.mQueue; //得到当前looper的MQ
    Binder.clearCallingIdentity();
    finallong ident = Binder.clearCallingIdentity();
    while (true) {
        Message msg = queue.next(); // 取出message
        if (msg != null) {
            if (msg.target == null) { return; }
            if (me.mLogging != null)
                me.mLogging.println(">>>> Dispatching to " + msg.target + " "
                                   + msg.callback + ": " + msg.what);
            // 将处理工作交给message的target, 即后面要讲的handler
            msg.target.dispatchMessage(msg);
            if (me.mLogging != null) me.mLogging.println("<<<< Finished to "
                                                       + msg.target + " " + msg.callback);
            finallong newIdent = Binder.clearCallingIdentity();
            if (ident != newIdent) {
                Log.wtf("Looper", "Thread identity changed from 0x"
                       + Long.toHexString(ident)
                       + " to 0x" + Long.toHexString(newIdent) + " while dispatching to "
                       + msg.target.getClass().getName() + " "
                       + msg.callback + " what=" + msg.what);
            }
            msg.recycle(); // 回收message资源
        }
    }
}
...
}

```

在系统的MediaPlayerActivity.Java中，我们可以看一下再OnCreate中的有这样的两句：

```
mAlbumArtWorker = new Worker("album art worker");  
mAlbumArtHandler = new AlbumArtHandler(mAlbumArtWorker.getLooper());
```

```
private class Worker implements Runnable {  
    private final Object mLock = new Object();  
    private Looper mLooper;  
    Worker(String name) {  
        Thread t = new Thread(null, this, name);  
        t.setPriority(Thread.MIN_PRIORITY);  
        t.start();  
        synchronized (mLock) {  
            while (mLooper == null) {  
                try {  
                    mLock.wait();  
                } catch (InterruptedException ex) {  
                }  
            }  
        }  
    }  
    public Looper getLooper() { return mLooper; }  
    public void run() {  
        synchronized (mLock) {  
            Looper.prepare();  
            mLooper = Looper.myLooper();  
            mLock.notifyAll();  
        }  
        Looper.loop();  
    }  
    public void quit() { mLooper.quit(); }  
}
```