Android 应用软件设计

E 5　IPC

学号：SA18225402
姓名：吴文韬
报告撰写时间：2018/10/6

一、 主要目标

1. 修改 E4 的 SCOS 包结构，增加 es.source.code.service 子包
2. 在 es.source.code.service 子包中新建类 ServerObserverService，继承 Service 父类，并重写父类方法 onBind()，要求如下：
   1) 在 ServerObserverService 类中新建 Handler 对象 cMessageHandler，重写方法 handleMessage()；当传入的 Message 属性 what 值为 1 时：启动多线程模拟接收服务器传回菜品库存信息（菜名称，库存量），每次接收到新数据，判断 SCOS app 进程是否在运行状态,如果为运行状态则向 SCOS app 进程发送 Message，其 what 值为 10，并在该 Message 中携带收到的库存信息（菜名称，库存量），多线程休眠频率为 300ms
   2) 当对象 cMessageHandler 使用 handleMessage()方法收到 Message 属性 what 值为 0 时，则关闭模拟接收服务器传回菜品库存信息的多线程
   3) 在 SCOS 的配置文件 AndroidManifest.xml 中注册 ServerObserverService 服务组件，并将其属性 android:process 设为 es.source.code.observerservice
3. 修改 E4 中的 FoodView 代码，将食物列表 ListView 中每个菜项信息增加一个 TextView，使用粗体蓝色字体显示当前该菜品库存量
4. 修改 E4 中的 FoodView 代码,将当前屏幕 ActionBar 菜单项增加新操作为“启动实时更新”，当用户点击该 Action 时：
   1) 启动 ServerObserverService 服务，并向 ServerObserverService 发送信息 Message 属性 what 值为 1
   2) 在 FoodView 中新建 sMessageHandler 对象，重写方法

handleMessage();当传入的Message属性what值为10时，解析该Message携带菜品库存信息（菜名称，库存量），并根据该值，更新FoodView菜项信息

3) “启动实时更新”Action被点击后，将Action标签修改为“停止实时更新”

4) 当在FoodView界面中点击“停止实时更新”Action时，向ServerObserverService发送Message属性what值为0

5. 在SCOS的es.source.code.service中新建类UpdateService，继承IntentService,并重写onHandleIntent()方法,要求如下：
   1) 模拟检查服务器是否有菜品种更新信息，如有更新则使用NotificationManager 发送状态栏通知，通知内容为“新品上架：菜名，价格，类型”
   2) 当点击该通知时，屏幕跳转至FoodDetailed界面，显示该菜品详细信息
   3) 在SCOS工程的AndroidManifest.xml注册该服务组件
6. 修改E4的SCOS包结构，增加es.source.code.br子包在es.source.code.br子包中新建类DeviceStartedListener，继承BroadcastReceiver父类，并重写onReceive()方法
   1) 当onReceive()方法接收到设备开机广播时，启动UpdateService服务
   2) 在SCOS工程的AndroidManifest.xml注册该广播接收器组件
7. 请根据以上任务自主添加数据结构或功能代码，测试功能实现
8. 请将SCOS工程中使用Handler和Message进行消息处理的代码切换至EventBus，并测试成功
9. 修改SCOS工程的AndroidManifest.xml中versionCode和versionName为3.0，打包发布SCOS3.0.apk，并用真机测试运行

二、 实现和证明

1，2.在新建的类中新建一个 Handler 对象实现对接受到的数据进行分析，当 what = 1 时，启动线程进行发送菜品的库存量。为 0 时取消线程。线程在服务创建时就开始运行，使用一个布尔标记来判断是否需要发送菜品库存信息。

```
private Handler cMessageHandler = new Handler(){
    @Override
    public void handleMessage(Message msg) {
        super.handleMessage(msg);
        if(msg.what == 1){
            //开启服务器线程
            if(!threadRunningTag){
                threadRunningTag = true;
            }
        }else if(msg.what == 0){
            //关闭服务器线程
            threadRunningTag = false;
        }else if(msg.what == MSG_BIND){
            //获得绑定的Activity的messenger
            activityMessenger = msg.replyTo;
        }
    }
};
```

```
@Override
public void onCreate() {
    super.onCreate();
    ServiceRunningTag = true;
    threadRun();
}
```

要注意的是在 activity 和 service 中使用 handle 通信时需要分别获得对方的 handle 才能实现双向通信，这里使用了绑定服务来获得双方的 handle 实现通信。

```
@Override
public IBinder onBind(Intent intent) {
    Messenger messenger = new Messenger(cMessageHandler);
    return messenger.getBinder();
}
```

```java
/**
 * 接口ServiceConnection内容
 * @param name
 * @param service
 */
@Override
public void onServiceConnected(ComponentName name, IBinder service) {
    //获得service的messenger
    serviceMessenger = new Messenger(service);

    //创建Activity的messenger
    Messenger messenger = new Messenger(mHandler);

    //创建消息
    Message msg = new Message();
    msg.what = ServerObserverService.MSG_BIND;
    msg.replyTo = messenger;

    //使用服务方的msg来发送本activity的messenger
    try {
        //在Service中就可以接收到Activity中的Messenger  用于Service与Activity之间的双向通信
        serviceMessenger.send(msg);
    } catch (RemoteException e) {
        e.printStackTrace();
    }
}
```

```java
/**
 * 启动模拟服务器发送数据进程
 */
private void  threadRun(){
    new Thread(new Runnable() {
        @Override
        public void run() {
            while(ServiceRunningTag){

                if(threadRunningTag) {
                    //四种类型的菜中随机选择一个
                    int number = mRandom.nextInt( bound: 4) + 1;

                    //在选中的菜品中随机选一个位置
                    int pos = mRandom.nextInt(FoodItems.getName(number).length);

                    //随机生成一个库存量
                    int storage = mRandom.nextInt( bound: 30);

                    Message message = new Message();

                    //打包数据
                    Bundle bundle = new Bundle();
                    bundle.putInt("number", number);
                    bundle.putInt("pos", pos);
                    bundle.putInt("storage", storage);

                    message.setData(bundle);
                    message.what = 10;

                    //发送数据
                    try {
                        activityMessenger.send(message);
                    } catch (RemoteException e) {
                        e.printStackTrace();
                    }
                }
                //休眠300ms
                try {
                    Thread.sleep( millis: 300);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }).start();
}
```

并且在 Manifest 文件中注册 service

```
<service
    android:name="es.source.code.service.ServerObserverService"
    android:enabled="true"
    android:exported="true"
    android:process="es.source.code.observerservice">
</service>
```

3. 在 FoodView 中新增了一个 TextView 为库存量，并且在 FoodItem 类中为每个菜品新增了一个 int 数组用于存放库存量，在初始化列表时使用该数组进行初始化。

```
public static String[] hot_food_name = {"香辣牛肉西兰花","蚂蚁上树","家常干锅排骨","菠萝咕咾肉","家常口水鸡","地三鲜","干贝边炖豆腐","鱼香肉丝","水煮肉片","铁锅芋头鸡"};
public static int[] hot_food_image = {R.drawable.hot_food_1,R.drawable.hot_food_2,R.drawable.hot_food_3,R.drawable.hot_food_4,R.drawable.hot_food_5,
                                      R.drawable.hot_food_6,R.drawable.hot_food_7,R.drawable.hot_food_8,R.drawable.hot_food_9,R.drawable.hot_food_10};
public static int[] hot_food_price = {35,67,89,58,66,52,32,55,76,73};
public static int[] hot_food_storage = {18,23,22,31,25,12,23,21,7,12};
```

4. 在 FoodView 的 ActionBar 中的 menu 文件中添加一个 item，在 onOptionsItemSelected 方法中实现对该 item 的点击相应操作。发消息时使用 service 获得的 messenger 发送。

```
case R.id.food_view_startService: {
    if(item.getTitle().equals("启动实时更新")) {
        item.setTitle("停止实时更新");

        //新建消息
        Message message = new Message();
        message.what = 1;
        //使用service的messenger发送消息
        try {
            serviceMessenger.send(message);
        } catch (RemoteException e) {
            e.printStackTrace();
        }

    }else if(item.getTitle().equals("停止实时更新")){
        item.setTitle("启动实时更新");

        //新建消息
        Message message = new Message();
        message.what = 0;
        //使用service的messenger发送消息
        try {
            serviceMessenger.send(message);
        } catch (RemoteException e) {
            e.printStackTrace();
        }

    }
    return true;
```

在 handle 接受到数据时，进行数据处理，并且对 listview 实现更新数据。更新数据操作如下所示。

```
private Handler mHandler = new Handler(){
    @Override
    public void handleMessage(Message msg) {
        super.handleMessage(msg);
        if(msg.what == 10){
            //处理从服务端收到的message
            Bundle bundle = msg.getData();
            int number = bundle.getInt( key: "number", defaultValue: 0);
            int pos = bundle.getInt( key: "pos");
            int storage = bundle.getInt( key: "storage");
            switch (number){
                case FoodItems.HOT_FOOD:
                    FoodItems.hot_food_storage[pos] = storage;
                    hotFoodAdapter.notifyDataSetChanged();
                    break;
                case FoodItems.COLD_FOOD:
                    FoodItems.cold_food_storage[pos] = storage;
                    coldFoodAdapter.notifyDataSetChanged();
                    break;
                case FoodItems.SEE_FOOD:
                    FoodItems.see_food_storage[pos] = storage;
                    seeFoodAdapter.notifyDataSetChanged();
                    break;
                case FoodItems.DRINK:
                    FoodItems.drink_storage[pos] = storage;
                    drinkAdapter.notifyDataSetChanged();
                    break;
                default:
            }
        }
    }
};
```

5. 新建 UpdateService 类，在类中的 onHandleIntent 方法添加处理代码。

```
@Override
protected void onHandleIntent(@Nullable Intent intent) {
    //获取intent中的数据
    Bundle bundle = intent.getBundleExtra( name: "message");
    String tag = bundle.getString( key: "tag");
    if(tag == null){
        //发送接收到库存更新信息
        sendFoodMessage(bundle);
    }else{
        //发送接收到开机启动完成信息
        sendBootStartMessage();
    }
}
```

```
private void sendBootStartMessage(){
    Intent clickIntent = new Intent( packageContext: UpdateService.this, SCOSEntry.class);
    PendingIntent contentIntent = PendingIntent.getActivity( context: UpdateService.this, "SCOS",clickIntent,PendingIntent.FLAG_UPDATE_CURRENT);

    //初始化通知栏信息
    Notification.Builder mBuilder = new Notification.Builder( context: UpdateService.this);
    mBuilder.setAutoCancel(true);
    mBuilder.setSmallIcon(R.drawable.logo);
    mBuilder.setTicker("欢迎点菜！");
    mBuilder.setWhen(System.currentTimeMillis());
    mBuilder.setLargeIcon(BitmapFactory.decodeResource(getResources(),R.drawable.logo));
    mBuilder.setContentTitle("欢迎点菜！");
    mBuilder.setContentIntent(contentIntent);
    mBuilder.setContentText("欢迎打开SCOS进行点菜");

    //安卓8.0后的通知需要设置渠道否则不显示
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        mBuilder.setChannelId("SCOS");
    }

    Notification notification = mBuilder.build();

    //初始化通知管理并且发送通知
    NotificationManager mNotificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
    mNotificationManager.notify("SCOS",notification);
}
```

```
private void sendFoodMessage(Bundle bundle){
    int number = bundle.getInt( key: "number");
    int pos = bundle.getInt( key: "pos");
    int storage = bundle.getInt( key: "storage");

    String foodName;
    int price;
    //初始化点击的intent数据
    Intent clickIntent = new Intent( packageContext: UpdateService.this, FoodDetailed.class);
    String name;
    switch (number){
        case FoodItems.HOT_FOOD:name = "hotFood";foodName = FoodItems.hot_food_name[pos];price = FoodItems.hot_food_price[pos];break;
        case FoodItems.COLD_FOOD:name = "coldFood";foodName = FoodItems.cold_food_name[pos];price = FoodItems.cold_food_price[pos];break;
        case FoodItems.SEE_FOOD:name = "seeFood";foodName = FoodItems.see_food_name[pos];price = FoodItems.see_food_price[pos];break;
        default:name = "drink";foodName = FoodItems.drink_name[pos];price = FoodItems.drink_price[pos];break;
    }
    clickIntent.putExtra( name: "data",name);
    clickIntent.putExtra( name: "pos",pos);

    PendingIntent contentIntent = PendingIntent.getActivity( context: UpdateService.this, "SCOS",clickIntent,PendingIntent.FLAG_UPDATE_CURRENT);

    //初始化通知栏信息
    Notification.Builder mBuilder = new Notification.Builder( context: UpdateService.this);
    mBuilder.setAutoCancel(true);
    mBuilder.setSmallIcon(R.drawable.logo);
    mBuilder.setTicker("新品上架！");
    mBuilder.setWhen(System.currentTimeMillis());
    mBuilder.setLargeIcon(BitmapFactory.decodeResource(getResources(),R.drawable.logo));
    mBuilder.setContentTitle("新品上架！");
    mBuilder.setContentIntent(contentIntent);
    mBuilder.setContentText("类型:"+name +" 菜名: "+foodName+" 价格: "+price);

    //安卓8.0后的通知需要设置渠道否则不显示
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        mBuilder.setChannelId("SCOS");
    }

    Notification notification = mBuilder.build();

    //初始化通知管理并且发送通知
    NotificationManager mNotificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
    mNotificationManager.notify("SCOS",notification);
}
```

使用 mBuilder 来创建一个 notification 消息，通过 notificationManager 来发送消息，要注意的是必须使用 setSmallIcon 来对要发送的消息进行设置，否则将不会看到要发送的消息。还要注意的是在安卓 8.0 系统中，对消息的设置中增加了渠道设置，如果是 8.0 系统则必须使用 setChannelId 进行设置，否则也会看不到设置的消息。

在 Manifest 文件中对 service 进行配置

```
<service
    android:name="es.source.code.service.UpdateService"
    android:exported="true"
    android:enabled="true">
    <intent-filter>
        <action android:name="es.source.code.service.openUpdateService"/>
    </intent-filter>
</service>
```

6.新建 DeviceStartedListener 类，并且重载 onReceive 方法。如果接收到的广播是我们要监听的系统启动完成广播，则执行相应的代码。

```
@Override
public void onReceive(Context context, Intent intent) {

    if(intent.getAction().equals("android.intent.action.BOOT_COMPLETED")){
        Intent serviceIntent = new Intent( action: "es.source.code.service.openUpdateService");
        serviceIntent.putExtra( name: "tag", value: "startSCOSEntry");
        context.startService(serviceIntent);
    }
}
```

在 Manifest 文件中对广播进行注册，并且声明需要的相应权限。

```
<receiver
    android:name="es.source.code.br.DeviceStartedListener"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>

</receiver>
```

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

需要注意的是在许多手机上都会自动被一些安全助手关闭自动启动功能，若要自动启动还需要手动开启功能。

7，8.首先在 gradle 文件中添加依赖包

```
    implementation 'org.greenrobot:eventbus:3.1.1'
```

在 SCOSHelper 中的 onCreate()方法中订阅 EventBus,并且 onDestroy()方法中取消订阅 EvenetBus.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.scos_helper);

    mGridView = findViewById(R.id.scos_helper_grid);

    initGridView();

    //订阅EventBus
    EventBus.getDefault().register( subscriber: this);
}
```

```
@Override
protected void onDestroy() {
    super.onDestroy();

    //取消订阅EventBus
    EventBus.getDefault().unregister( subscriber: this);
}
```

在实现对信息的处理之前，先定义了一个自定义类 EveneBusMessage 用来实现信息的创建和接收。

```
/**
 * 实现EventBus的类
 */
public class EventBusMessage {

    //传递的消息
    private String message;

    public EventBusMessage(String message) { this.message = message; }

    public String getMessage() { return message; }

}
```

之后在 SCOSHelper 中新建一个 onEvenet 方法，并且为其加上 @Subscribe，同时由于要在该方法中更新 UI，所以使用的是 MAIN。

```
/**
 * 处理收到的eventBusMessage
 * @param eventBusMessage
 */
@Subscribe(threadMode = ThreadMode.MAIN)
public void onEvent(EventBusMessage eventBusMessage){
    if(eventBusMessage.getMessage().equals("1")){
        Toast.makeText( context: SCOSHelper.this, text: "邮件发送成功",Toast.LENGTH_SHORT).show();
    }else if(eventBusMessage.getMessage().equals("-1")){
        Toast.makeText( context: SCOSHelper.this, text: "正在发送邮件...",Toast.LENGTH_SHORT).show();
        Toast.makeText( context: SCOSHelper.this, text: "邮件发送中，请耐心等待",Toast.LENGTH_LONG).show();
    }else{
        Toast.makeText( context: SCOSHelper.this, text: "邮件发送失败",Toast.LENGTH_SHORT).show();
    }
}
```

并且在之前使用 handle 发送消息的地方均更改为使用 EventBus 发送消息。

```
    /**
     * 处理邮件帮助
     */
    private void emailHelp(){
        new Thread(new Runnable() {
            @Override
            public void run() {
                EventBus.getDefault().post(new EventBusMessage("-1"));
//                mHandler.sendEmptyMessage(-1);
                try {
                    sendEmail();
                    EventBus.getDefault().post(new EventBusMessage("1"));
//                    mHandler.sendEmptyMessage(1);
                } catch (Exception e) {
                    e.printStackTrace();
                    EventBus.getDefault().post(new EventBusMessage("0"));
//                    mHandler.sendEmptyMessage(0);
                }

            }
        }).start();
    }

    /**
```

9. 修改 gradle 文件实现发布新版本

```
versionCode 4
versionName "3"
```

三、 结论

1. 为了实现 activity 和 service 的双向通信，这里我们使用了绑定的方法启动了 service，使用绑定的好处就是可以使 activity 获得 service 的 messenger，使 service 获得 activity 的 messenger，这样就能使这两个组件实现双向通信，分别在两个组件中各自实现收到信息时应当做出的反应。
2. 在 listview 中的 adpter 信息发生变化时，不应该重新初始化来刷新界面，应该使用 notifyDataSetChanged 方法来单个刷新变化的 item
3. 在向通知栏发送消息时，消息必须是使用过 setSmallIcon 方法来设置过的不然通知栏不会显示通知，而且在安卓 8.0 版本以后，新增了一个渠道信息，必须使用 setChannelId 方法设置渠道否则在 8.0 版本以后也不会显示通知。
4. 在广播中若要接收系统启动完成的广播，必须在 Manifest 文件中先声明 user-permission 才行，并且在广播的过滤器中增加对这个广播的监听。
5. 使用 EventBus 前需要添加对其的依赖，因为他是第三方的包。并且需要在使用的 activity 的 onCcreate 方法中注册，在 onDestroy 方法中取消注册。

四、 参考文献

1. 使用 Handler 实现 Service 和 Activity 之间的双向通信
2. 解决 Android 8.0 的 Notification 不显示问题
3. Android 8.0 通知
4. Android 开机启动 APP 广播