Android 应用软件设计

E 5　IPC

学号：SA17225268
姓名：彭明

报告撰写时间：2017/10/31

1.主题概述

本次主题是进程间通信，是指两个进程之间进行数据交换的过程。这次进程间通信使用到了 Messenger，通过它可以在不同进程中传递 Message 对象，在 Message 中放入我们需要传递的数据，就可以轻松实现数据的进程间传递了。

2.假设

主题内容假设可以请求服务器数据。

3.实现或证明

1. 增加 es.source.code.service 子包，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()

```
public class ServerObserverService extends Service {
    //线程停止标志
    private Boolean exit = false;
    private Messenger cMessenger;

    private Handler cMessageHandler = new Handler(){
        @Override
        public void handleMessage(final Message msg) {
            cMessenger = msg.replyTo;
            switch (msg.what){
                case 0:
                    exit = true;
                    break;

                case 1:
                    exit = false;

                    new Thread(new Runnable() {
                        @Override
                        public void run() {
                            while (!exit){
                                Food food = null;

                                try{
                                    //模拟请求服务器数据
                                    food = new Food("凉拌海带丝", 18, "", "",
                                            "冷菜", 20, 0, false);
                                    Thread.sleep(300);
                                } catch (Exception e) {
```

```
                                e.printStackTrace();
                            }

                            if(food != null && isAppRunning()){
                                Message message = Message.obtain();
                                message.what = 10;
                                Bundle bundle = new Bundle();
                                bundle.putSerializable("Food", food);
                                message.setData(bundle);

                                try{
                                    cMessenger.send(message);
                                } catch (RemoteException e) {
                                    e.printStackTrace();
                                }
                            }
                        }
                    }
                }).start();
                break;
            default:
        }
    }
};

//判断 SCOS app 进程是否在运行状态
public Boolean isAppRunning(){
    ActivityManager am = (ActivityManager)
getSystemService(Context.ACTIVITY_SERVICE);
    List<ActivityManager.RunningAppProcessInfo> list =
am.getRunningAppProcesses();
    if(list == null){
        return false;
    }
    for(ActivityManager.RunningAppProcessInfo processInfo : list){
```

```
            if(processInfo.processName.equals(getPackageName())
                    && processInfo.importance ==
ActivityManager.RunningAppProcessInfo.IMPORTANCE_FOREGROUND){
                return true;
            }
        }
        return false;
    }


    private Messenger sMessenger = new Messenger(cMessageHandler);

    public ServerObserverService() {
    }

    @Override
    public IBinder onBind(Intent intent) {
        return sMessenger.getBinder();
    }
}
```

2. 在 SCOS 的配置文件 AndroidManifest.xml 中注册 ServerObserverService 服务组件，并将其属性“android:process”定义为“es.source.code.observerservice”

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

3. 修改 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

在 layout 文件中新增一个 TextView

```
<TextView
        android:id="@+id/food_number"
        android:layout_gravity="center"
        android:layout_width="wrap_content"
        android:layout_weight="2"
        android:textStyle="bold"
        android:textColor="@color/colorPrimary"
        android:layout_height="wrap_content" />
```

在 FoodRvAdapter 中绑定 TextView

```
        holder.foodNumber.setText(String.valueOf(foods.get(position).
getNumber()));
```

4. 修改 FoodView 代码，将当前屏幕 ActionBar 菜单项增加新操作为“启动实时更新”

在 onCreate 中绑定 service

```
Intent service = new Intent(FoodView.this, ServerObserverService.class);
bindService(service, serviceConnection, BIND_AUTO_CREATE);
```

在 onDestroy 中取消绑定

```
unbindService(serviceConnection);
```

在 serviceConnection 中获取到 sMessenger

```
    private Messenger sMessenger;

    private ServiceConnection serviceConnection = new ServiceConnection()
{
        @Override
```

```
        public void onServiceConnected(ComponentName name, IBinder
service) {
            sMessenger = new Messenger(service);
        }

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

通过 sMessageHandler 来构造 cMessenger

```
private Handler sMessageHandler = new Handler(){
        @Override
        public void handleMessage(Message msg) {
            switch (msg.what){
                case 10:
                    Food food = (Food)
msg.getData().getSerializable("Food");

                    int index;
                    switch (food.getKind()){
                        case "冷菜":
                            index = 0;
                            break;
                        case "热菜":
                            index = 1;
                            break;
                        case "海鲜":
                            index = 2;
                            break;
                        case "酒水":
                            index = 3;
                            break;
                        default:
```

```
                }

                //获取到 fragment 的引用来调用 fragment 中的 dataChanged 方法更新 FoodView 菜品信息
                PlaceholderFragment fragment = (PlaceholderFragment) getSupportFragmentManager().findFragmentByTag("android:switcher:"+R.id.container_food+":"+index);
                if(fragment != null){
                    fragment.dataChanged(food);
                }
                break;

            default:
        }
    }
};

private Messenger cMessenger = new Messenger(sMessageHandler);
```

在 onOptionsItemSelected 中处理启动实时更新点击事件

```
private static final String START_ASYNCSERVICE = "启动实时更新";

case R.id.start_asyncservice:
    if(START_ASYNCSERVICE.equals(item.getTitle())){
        item.setTitle(R.string.stop_asyncservice);
        Message message = Message.obtain();
        message.what = 1;
        message.replyTo = cMessenger;

        try{
            sMessenger.send(message);
        } catch (RemoteException e) {
            e.printStackTrace();
        }
```

```
            } else {
                item.setTitle(R.string.start_asyncservice);
                Message message = Message.obtain();
                message.what = 0;
                message.replyTo = cMessenger;

                try{
                    sMessenger.send(message);
                } catch (RemoteException e) {
                    e.printStackTrace();
                }
            }
            break;
```

5. 在 SCOS 的 es.source.code.service 中新建类 UpdateService，继承 IntentService，并重写 onHandleIntent()方法

```
public class UpdateService extends IntentService {

    private static final String CHANNEL_ID = "es.source.code.service";
    private static final String CHANNEL_NAME = "DEFAULT CHANNEL";

    private ArrayList<Food> foods;

    public UpdateService() {
        super("UpdateService");
    }

    @Override
    protected void onHandleIntent(@Nullable Intent intent) {

        foods = new ArrayList<Food>();

        //模拟检查服务器菜品种更新信息
```

```
        Food fd = new Food("凉拌海带丝", 18, "", "",
                "冷菜", 20, 0, false);

        foods.add(fd);

        if(foods != null){
            Intent detailIntent = new Intent(UpdateService.this,
FoodDetailed.class);
            detailIntent.putExtra("foods", foods);
            detailIntent.putExtra("position", 0);
            PendingIntent pi =
PendingIntent.getActivity(UpdateService.this, 0, detailIntent, 0);


            NotificationManager manager = (NotificationManager)
getSystemService(NOTIFICATION_SERVICE);
            NotificationCompat.Builder notificationCompatBuilder;

            if(Build.VERSION.SDK_INT < 26){
                notificationCompatBuilder = new
NotificationCompat.Builder(UpdateService.this);
            } else {
                NotificationChannel channel = new
NotificationChannel(CHANNEL_ID, CHANNEL_NAME,
NotificationManager.IMPORTANCE_DEFAULT);
                channel.enableLights(true);
                channel.enableVibration(true);
                channel.setLightColor(Color.GREEN);

channel.setLockscreenVisibility(Notification.VISIBILITY_PRIVATE);
                manager.createNotificationChannel(channel);
                notificationCompatBuilder = new
NotificationCompat.Builder(UpdateService.this, CHANNEL_ID);
            }
```

```
            Food food = foods.get(0);
            Notification notification = notificationCompatBuilder
                    .setContentTitle("新品上架")
                    .setContentText("菜名:" + food.getFoodName() + " " + "
价格:" + "¥" + food.getFoodPrice() + " " + "类型:" + food.getKind())
                    .setWhen(System.currentTimeMillis())
                    .setSmallIcon(R.mipmap.ic_launcher)
                    .setLargeIcon(BitmapFactory.decodeResource(getResour
ces(), R.drawable.socs_launcher))
                    .setContentIntent(pi)
                    .build();

            manager.notify(1, notification);
        }
    }
}
```

6. 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

   当 onReceive()方法接收到设备开机广播时，启动 UpdateService 服务

```
    @Override
    public void onReceive(Context context, Intent intent) {
        Intent updateService = new Intent(context, UpdateService.class);
        context.startService(updateService);
    }
```

在 SCOS 工程的 AndroidManifest.xml 注册该广播接收器组件及添加相应权限

```
        <receiver
            android:name="es.source.code.br.DeviceStartedListener"
```

```
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action
android:name="android.intent.action.BOOT_COMPLETED" />
            </intent-filter>
        </receiver>
```

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

7. 将 SCOS 工程中使用 Handler 和 Message 进行消息处理的代码切换至 EventBus

首先有 EventBus 不能跨进程使用，需去掉 ServerObserverService 的 process 属性

在 FoodView 和 ServerObserverService 的 onCreate 和 onDestroy 中分别加入以下代码

```
EventBus.getDefault().register(this);
```

```
EventBus.getDefault().unregister(this);
```

新建 Msg 类来传递消息

```
public class Msg {
    private Boolean exit;
    public Msg(Boolean exit){
        this.exit = exit;
    }

    public Boolean getExit() {
        return exit;
    }

    public void setExit(Boolean exit) {
        this.exit = exit;
    }
}
```

在 FoodView 中加入以下代码，

启动时：

```
EventBus.getDefault().post(new Msg(false));
```

停止时：

```
EventBus.getDefault().post(new Msg(true));
```

接收 ServerObserverService 传过来的 Food

```
@Subscribe(threadMode = ThreadMode.MAIN)
       public void updateFoodView(Food event) {
           foods.add(event);
           foodAdapter.notifyDataSetChanged();
       }
```

在 ServerObserverService 中加入以下代码

```
@Subscribe(threadMode = ThreadMode.ASYNC)
    public void getFoodFromServer(Msg event){
        while (!event.getExit()){
            Food food = null;

            try{
                food = new Food("凉拌海带丝", 18, "", "",
                        "冷菜", 20, 0, false);
                Thread.sleep(300);
            } catch (Exception e) {
                e.printStackTrace();
            }

            if(food != null && isAppRunning()){
                EventBus.getDefault().post(food);
            }
```

```
        }
    }
```

4.结论

进程间通信可以很好地解决信息传递问题，而 Messenger 是一种实现进程间通信的方法。EventBus 是一种很方便地在线程间传递信息的工具。

5.参考文献

1. Android 官方文档中关于 Messenger 的介绍
   https://developer.android.google.cn/reference/android/os/Messenger.html

2. Android 的进阶学习--Messenger 的使用和理解
   http://www.jianshu.com/p/af8991c83fcb

3. 开机广播和 IntentService
   http://www.jianshu.com/p/378819c21bde

4. Android 官方文档中关于 Notification 的介绍
   https://developer.android.google.cn/reference/android/app/Notification.html

5. Github 上 EventBus 库
   https://github.com/greenrobot/EventBus