Android 应用软件设计

E 4　Multi-thread Programming

学号：SA18225402
姓名：吴文韬
报告撰写时间：2018/10/3

一、 主要目标

1. 修改 E 3 中的 LoginOrRegister 代码，当导航到该页面时，使用 SharedPreferences 保存或读取用户登录状态：

   1）判断 SharedPreferences 中是否有用户名（userName）记录。如果无表示用户未曾注册或登录过 SCOS，在 LoginOrRegister 界面中隐藏登录按钮，其他不变。

   2）如果 SharedPreferences 中有用户名记录，表示该用户已经注册或登录过 SCOS，在 LoginOrRegister 中隐藏注册按钮，并将登录名中默认显示 SharedPreferences 中的用户名值，其他不变。
2. 修改 LoginOrRegister 代码，当用户点击登录时，按 E3 中登录规则判断是否通过，如果登录成功则将当前登录名输入框的值写入到 SharedPreferences 中的 userName 记录中，另外再写入 loginState 值为 1，其他保持不变。
3. 修改 LoginOrRegister 代码，当用户点击注册时，将当前登录名输入框的值写入到 SharedPreferences 中的 userName 记录中，另外再写入 loginState 值为 1,其他保持不变。
4. 修改 LoginOrRegister 代码，当用户点击返回时，判断 SharedPreferences 中是否有用户名记录，如无则保持原功能不变,否则在 SharedPreferences 写入 loginState 值为 0。
5. 修改 MainScreen 和 SCOSEntry 代码，参照以上任务，将用户是否登录的判断逻辑改成使用 SharedPreferences 的 loginState 值进行功能实现。
6. 在 SCOS 的 es.source.code.activity 包中，新建 SCOSHelper.java 类为 Activity 子类。该 Activity 中设置 RelativeLayout 布局为：

   1）使用 GridView 显示该页面帮助选项：用户使用协议、关于系统、电话人工帮助、短信帮助、邮件帮助。

   2）当用户点击“电话人工帮助”时，使用 ImplicitIntent 调用系统 app 实现自动拨号功能，拨打目标号码为“5554”

   3）当用户点击“短信帮助”时，使用 SmsManager 自动发送短信至目标

号码“5554”，内容为“test scos helper”。当短信发送完毕后，使用 Toast 提示“求助短信发送成功”

4）当用户点击“邮件帮助”时，启动自定义多线程 MailSender 完成邮件发送，邮件内容和标题自定义。当邮件发送完毕后，使用 Handler 接收消息，当消息值为 1 时，使用 Toast 提示“求助邮件已发送成功。

7. 修改 E3 中 FoodOrderView 代码，使用 AsyncTask 模拟结账功能。当用户点击结账按钮时，启动 AsyncTask，并使用 ProgressBar 实时显示结账进度，要求 AsyncTask 运行 6 秒后模拟完成结账功能。当结账完成后，修改结账按钮为不可点击状态，并使用 Toast 提示用户本次结账金额及积分增加情况。
8. 总结 Android 应用开发中，如何提高程序性能和避免 ANR 异常的出现？
9. 在多线程编程中，怎样保证线程安全？

二、 实现和证明

1. 在 LoginOrRegister.java 中添加如下代码，实现当 userName 为空时，隐藏登录按钮，否则隐藏注册按钮并且把用户名放入用户名输入框中。代码如下所示。

```
mSharedPreferences = getSharedPreferences( name: "SCOSData", Context.MODE_PRIVATE);
mEditor = mSharedPreferences.edit();

//Toast.makeText(LoginOrRegister.this, "如需注册请先输入用户名密码在点击下方注册按钮",Toas

//判断是否已经登录过，若为未登录过则隐藏登录按钮
String userName = mSharedPreferences.getString("userName","");
if(userName.equals("")){
    login.setVisibility(View.INVISIBLE);
    login.setEnabled(false);
}else {
    register.setVisibility(View.INVISIBLE);
    register.setEnabled(false);
    user_name.setText(userName);
    user_name.setEnabled(false);
}
```

2. 在 LoginOrRegister.java 中的 Onclick 方法中对点击登录按钮的处理中添加如下代码，向 SharedPreferences 中写入用户名和登录状态。

```
//登录成功写入数据到SharedPreferences
mEditor.putString("userName",user_name.getText().toString());
mEditor.putInt("loginState",1);
```

3. 在 LoginOrRegister.java 中的 Onclick 方法中对点击注册按钮的处理中添加如下代码，向 SharedPreferences 中写入用户名和登录状态。

```
//注册成功写入数据到SharedPreferences
mEditor.putString("userName",user_name.getText().toString());
mEditor.putInt("loginState",1);
```

4. 在 LoginOrRegister.java 中的 Onclick 方法中对点击返回按钮的处理中添加如下代码，判断 SharedPreferences 中是否有用户名，若无则写入登录状态为 0。

```
//当返回按钮被按下时
String userName = mSharedPreferences.getString("userName","");
if(userName.equals("")){
    mEditor.putInt("loginState",0);
}
```

5. 只需要在 MainScreen.java 中中添加读取 SharedPreferences 中

loginState 的数据，根据数据来判断是否已经登录。

```
//根据SharedPreferences的loginState决定canBeVisible的值
SharedPreferences mSharedPreferences = getSharedPreferences( name: "SCOSData", Context.MODE_PRIVATE);
if(mSharedPreferences.getInt( key: "loginState", defValue: 0) == 1){
    canBeVisible = true;
}
```

并且应当在 onActivityResult()方法中修改为根据 loginState 的状态来决定是否应该初始化和显示点菜等按钮。

```
if(requestCode == 0){
        Bundle bundle = data != null ? data.getExtras() : null;
        if(bundle != null){
            String message = bundle.getString( key: "data");
            if(message.equals("RegisterSuccess")){
                Toast.makeText( context: MainScreen.this, text: "欢迎您成为SCOS新用户",Toast.LENGTH_SHORT).show();
            }
            int loginState = mSharedPreferences.getInt( key: "loginState", defValue: 0);
            if(loginState == 0){
                user = null;
                canBeVisible = false;
            }else if(loginState == 1){
                user = new User();
                canBeVisible = true;
            }
            setMStringsAndMImages();
        }
}else if(requestCode == 1){
```

6. 新建 SCOSHelper 后依照之前使用 GridView 的经验，创建好布局和适配是以及监听器后，对各个按钮进行处理。在处理之前需要在 Manifest 文件中添加如下权限：

```
<uses-permission android:name="android.permission.CALL_PHONE" />
<uses-permission android:name="android.permission.SEND_SMS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
```

1) 当点击电话人工帮助时，进入处理电话人工帮助方法中，考虑到安卓 6.0 以后一些权限需要动态分配，在打电话之前进行了权限的动态申请，当用户未授权时进入设置界面进行授权。如果有权限则直接进入打电话方法中。

```
if (ContextCompat.checkSelfPermission( context: this, Manifest.permission.CALL_PHONE) != PackageManager.PERMISSION_GRANTED) {
    if (ActivityCompat.shouldShowRequestPermissionRationale( activity: this, Manifest.permission.CALL_PHONE)) {

        Toast.makeText( context: this,  text: "请授权！", Toast.LENGTH_LONG).show();

        Intent intent = new Intent(Settings.ACTION_APPLICATION_DETAILS_SETTINGS);
        Uri uri = Uri.fromParts( scheme: "package", getPackageName(),  fragment: null);
        intent.setData(uri);
        startActivity(intent);

    } else {
        ActivityCompat.requestPermissions( activity: SCOSHelper.this, new String[]{Manifest.permission.CALL_PHONE}, MY_PERMISSIONS_REQUEST_CALL_PHONE);
    }
} else {
    makePhoneCall();
}
```

打电话方法的具体内容

```
private void makePhoneCall() {
    Toast.makeText( context: SCOSHelper.this,  text: "处理电话帮助", Toast.LENGTH_SHORT).show();
    Uri uri = Uri.parse("tel:5554");
    Intent intent = new Intent(Intent.ACTION_CALL, uri);
    if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.CALL_PHONE) != PackageManager.PERMISSION_GRANTED) {

        return;
    }
    startActivity(intent);
}
```

2）当点击短信按钮时同电话一样，先进行动态申请权限，在处理，具体代码如下

```
private void makePhoneCall() {
    Toast.makeText( context: SCOSHelper.this,  text: "处理电话帮助", Toast.LENGTH_SHORT).show();
    Uri uri = Uri.parse("tel:18651400987");
    Intent intent = new Intent(Intent.ACTION_CALL, uri);
    if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.CALL_PHONE) != PackageManager.PERMISSION_GRANTED) {

        return;
    }
    startActivity(intent);
}
```

```
/**
 * 发送短信
 */
private void sendMessage(){
    SmsManager smsManager = SmsManager.getDefault();

    PendingIntent sendIntent = PendingIntent.getBroadcast( context: this, requestCode: 0,new Intent( action: "SENT_SMS_ACTION"), flags: 0);

    registerReceiver(new BroadcastReceiver() {//注册广播接受短信发送是否成功的消息
        @Override
        public void onReceive(Context context, Intent intent) {
            if(getResultCode() == Activity.RESULT_OK){
                Toast.makeText( context: SCOSHelper.this, text: "短信发送成功",Toast.LENGTH_SHORT).show();
            }else{
                Toast.makeText( context: SCOSHelper.this, text: "短信发送失败",Toast.LENGTH_SHORT).show();
            }
        }
    },new IntentFilter( action: "SENT_SMS_ACTION"));

    smsManager.sendTextMessage( destinationAddress: "5554", scAddress: null, text: "test scos helper",sendIntent, deliveryIntent: null);

}
```

3）当用户点击邮件帮助时，由于发送邮件属于耗时操作，所以使用了线程来完成。首先定义了一个 Handler 来实现对线程发出的消息进行处理。

```
//处理接收邮件发送是否成功的信息
@SuppressLint("HandlerLeak")
private Handler mHandler = new Handler(){
    @Override
    public void handleMessage(Message msg) {
        super.handleMessage(msg);
        if(msg.what == 1){
            Toast.makeText( context: SCOSHelper.this, text: "邮件发送成功",Toast.LENGTH_SHORT).show();
        }else if(msg.what == -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();

        }
    }
};
```

在点击了邮件帮助后。进入如下代码，该代码定义了一个线程用来实现发送邮件的任务。

```
/**
 * 处理邮件帮助
 */
private void emailHelp(){
    new Thread(new Runnable() {
        @Override
        public void run() {

            mHandler.sendEmptyMessage( what: -1);
            try {
                sendEmail();
                mHandler.sendEmptyMessage( what: 1);
            } catch (Exception e) {
                e.printStackTrace();
                mHandler.sendEmptyMessage( what: 0);
            }

        }
    }).start();
}
```

在完成发送邮件的方法中，自定义了一个 class 用来简化发送邮件的代码，该类的内容如下，同时也使用了第三方的 jar 库，activation.jar,additioinnal.jar,mail.jar。类的内容及调用类的代码如下所示。

```
/**
 * 发送邮件
 */
private void sendEmail() throws Exception {

    MailSender sender = new MailSender( user: "18651400987@163.com",  password: "winter1996");
    sender.sendMail( subject: "SCOSHelper",  body: "There is some trouble when using scos!", sender: "18651400987@163.com",  recipients: "1012179010@qq.com");

}
```

```java
class  MailSender extends Authenticator {
    //服务器地址
    private String mailhost = "smtp.163.com";
    //用于发送邮件的邮箱地址
    private String user;

    //用于发送邮件的邮箱密码
    private String password;

    //会话（每创建一次连接就要有一个会话）
    private Session session;


    public MailSender(String user, String password) {
        this.user = user;
        this.password = password;

        //创建连接属性
        Properties props = new Properties();

        //设置通信协议
        props.setProperty("mail.transport.protocol", "SMTP");

        //设置服务器地址
        props.setProperty("mail.smtp.host", mailhost);

        //设置是否需要SMTP身份验证
        props.put("mail.smtp.auth", "true");
        //设置SSL协议端口号
        props.put("mail.smtp.port", "25");


        session = Session.getInstance(props, authenticator: this);
    }
    protected PasswordAuthentication getPasswordAuthentication() {
        return new PasswordAuthentication(user, password);
    }
```

```java
    public synchronized void sendMail(String subject, String body, String sender, String recipients) throws Exception {
        try{
            //创建邮件体
            MimeMessage message = new MimeMessage(session);
            DataHandler handler = new DataHandler(new ByteArrayDataSource(body.getBytes(), type: "text/plain"));
            message.setSender(new InternetAddress(sender));
            message.setSubject(subject);
            message.setDataHandler(handler);
            if (recipients.indexOf(',') > 0)
                message.setRecipients(javax.mail.Message.RecipientType.TO, InternetAddress.parse(recipients));
            else
                message.setRecipient(javax.mail.Message.RecipientType.TO, new InternetAddress(recipients));
            message.saveChanges();

            try {
                session.setDebug(true);
                //创建连接
                Transport trans = session.getTransport( protocol: "smtp");
                //连接服务器
                trans.connect(mailhost,user, password);

                //发送消息
                trans.send(message);
            } catch (AuthenticationFailedException ae) {
                ae.printStackTrace();
            } catch (MessagingException mex) {
                mex.printStackTrace();
                Exception ex = null;
                if ((ex = mex.getNextException()) != null) {
                    ex.printStackTrace();
                }
            }
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

7. 我在 FoodOrderView.java 中新建了一个内部类 MyAsyncTask 继承了 AsyncTask<String,Integer,String> 类，重载了 AsyncTask 中的 onPreExecute，onPostExecute，onProgressUpdate，onCancelled，doInBackground 等方法。具体代码如下：

```
class MyAsyncTask extends AsyncTask<String,Integer,String>{

    private String title;
    public MyAsyncTask(String title){
        super();
        this.title = title;
    }
    @Override
    protected void onPreExecute() {
        Toast.makeText( context: FoodOrderView.this, text: "正在支付中，请稍等...",Toast.LENGTH_SHORT).show();
        if(mProgressDialog == null|| !mProgressDialog.isShowing()){
            mProgressDialog = new ProgressDialog( context: FoodOrderView.this);
            mProgressDialog.setTitle("请稍等");
            mProgressDialog.setMessage(title+"...");
            mProgressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
            mProgressDialog.show();
        }
    }
    @Override
    protected void onPostExecute(String s) {
        Toast.makeText( context: FoodOrderView.this, text: "支付完成",Toast.LENGTH_SHORT).show();
        closeDialog();
    }
    @Override
    protected void onProgressUpdate(Integer... values) {
        mProgressDialog.setProgress(values[0]);
        mProgressDialog.setSecondaryProgress(0);
    }
    @Override
    protected void onCancelled(String s) {
        System.out.println("=====================================");
        Toast.makeText( context: FoodOrderView.this, text: "支付中断",Toast.LENGTH_SHORT).show();
        closeDialog();
    }
    @Override
    protected String doInBackground(String... strings) {
        int ratio = 0;
        for(;ratio<=100;ratio+=5){
            if(!mProgressDialog.isShowing()){
                cancel( mayInterruptIfRunning: true);
                return strings[0];
            }
            try {
                Thread.sleep( millis: 300);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            publishProgress(ratio);
        }

        return strings[0];
    }
```

并且在点击了结账按钮以后判断是否有菜品，无菜品提示先点菜，有菜品则进入点菜界面。

```
/**
 * 结账处理
 */
private void  settleAccount(){
    if(orderListView.getCount() == 0){
        Toast.makeText( context: FoodOrderView.this, text: "无菜品需要结账",Toast.LENGTH_SHORT).show();
    }else{
        MyAsyncTask myAsyncTask = new MyAsyncTask( title: "支付中");
        myAsyncTask.execute("支付中");
    }
}

/**
```

8. 在编写程序的时候，我们应当将复杂的计算和耗时间的操作放到子进程中进行，用来避免主程序过长时间无响应产生的 ANR 异常，以此来提高程序的性能。
9. 在多线程编程时，使用 synchronized 对共享资源进行加锁，从而实现线程安全。

三、 结论

1. 学会了简单的使 SharedPreferences 存储数据，用来实现在应用退出后保存数据，要注意的是 SharedPreferences 中 edit 数据后一定要提交 commit()，不然不会存储在文件中。
2. 学会了如何调用短信和电话等系统程序，调用这些程序之前需要在 Manifest 文件中先添加 user-permission，并且在 android6.0 之后需要在代码中添加动态申请的代码。
3. 在其他线程中不能对 UI 进行更新，可以使用 Handle 来进行消息的接收，通过消息来实现响应的操作。
4. 学会了使用多线程来处理耗时操作，例如发邮件和结账等操作。

四、 参考文献

1. Android Studio————权限获取
2. Android6.0 发送短信 Demo
3. Android 之消息处理——Handler
4. Android 后台发送邮件