3 注册模块

> “
>
> - 在开始阅读下面的内容前，请允许小编说几句题外话。由于同时还有其他事务要处理（现在也是），前一版的Lab Manual是在赶工的情况下完成的，没来得及仔细校验，给大家造成了很多困扰，小编也深刻感受到辜负了大家的学习热情，所以现在发布一个修订版本。
> - 蓦然的API文档不是小编写的，服务器端的处理逻辑是怎样的小编也只能一个个去尝试。昨天和其他同事确认了下，现在服务器成功处理的请求是能正常返回的，但是处理失败的消息却不像API文档里描述的那样。比如用已经注册过的邮箱再次注册时，并没有返回文档里描述的“Email exists”消息，但是有一段时间是没有问题的。如果服务器端的问题还是难以解决，慢慢地可能会由小编或其他同事接手服务器端的项目。
> - 某个步骤摘抄出来的代码，小编会顺带把这部分上面或下面的代码摘取少量一起贴出来，作为提示代码位置的上下文。大家见到上一版本某个Button的声明出现在两段摘抄出来的代码里，那个就是用来提示位置的，不是声明了两次，同一个对象在一个类里面只能声明一次，否则编译器会报错，无法运行的。
> - 最后，大家阅读文档时一定要参照着上下文来看，这样有前后矛盾的地方能大概判断出来哪个说法是正确的，不致受到误导。有时候即使已经反反复复地校验很多遍了，还是无法保证完全没有错误，这可能是由于小编知识结构的欠缺、记忆的混乱或错漏，或者只是因为自己写的文档，太熟悉反而发现不了错误等，但是现阶段只能由小编自己校验了。当然，有错误大家也直接提出来，这一系列Lab Manual我们会不断迭代更新。

这一章我们将带领大家在网络环境下完成在蓦然服务器的注册请求，主要包含以下内容：

- 界面跳转
- MD5算法
- 多线程
- 网络编程

3.1 用户界面的跳转

（1）完成界面布局

注册界面的布局和上一章的登录大同小异，只是多了昵称和重复密码两个输入框，除此之外，我们还可以添加一个进度条或进度对话框。

在`java/应用包名/ui/activity`目录下新建一个Activity，命名为`SignUpActivity`，参考页面布局如下：

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:background="@color/colorBackground"
    tools:context="com.geekband.demo.moran.ui.activity.SignUpActivity">

    <ProgressBar
        android:id="@+id/sign_up_progress"
        style="?android:attr/progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center|center_horizontal|center_vertical"
```

```
        android:layout_gravity="center|center_horizontal|center_vertical"
        android:visibility="gone"/>

    <ScrollView
        android:id="@+id/signin"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <LinearLayout
            android:id="@+id/signin_form"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="vertical">

            <FrameLayout
                android:layout_width="match_parent"
                android:layout_height="@dimen/logo_container_height"
                android:background="@color/colorOrange">

                <ImageView
                    android:layout_width="@dimen/logo_width"
                    android:layout_height="@dimen/logo_height"
                    android:src="@drawable/sign_logo"
                    android:layout_gravity="center|center_horizontal|center_vertical"/>
            </FrameLayout>

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:orientation="vertical"
                android:paddingLeft="@dimen/sign_vertical_padding"
                android:paddingRight="@dimen/sign_vertical_padding">


                <TextView
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_marginTop="18dp"
                    android:layout_marginBottom="@dimen/sign_vertical_margin"
                    android:paddingLeft="@dimen/sign_padding_left"
                    android:textSize="@dimen/sign_text_size"
                    android:textColor="@color/colorGrey"
                    android:text="@string/prompt_email"/>

                <AutoCompleteTextView
                    android:id="@+id/email"
                    android:inputType="textEmailAddress"
                    android:layout_width="match_parent"
                    android:layout_height="@dimen/sign_view_height"
                    android:paddingLeft="@dimen/sign_padding_left"
                    android:maxLines="1"
                    android:singleLine="true"
                    android:background="@drawable/text_shape"/>

                <TextView
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_marginTop="@dimen/sign_vertical_margin"
                    android:layout_marginBottom="@dimen/sign_vertical_margin"
                    android:paddingLeft="@dimen/sign_padding_left"
                    android:textSize="@dimen/sign_text_size"
                    android:text="@string/prompt_nickname"
                    android:textColor="@color/colorGrey"
                    />
                <EditText
                    android:id="@+id/nickname"
                    android:layout_width="match_parent"
                    android:layout_height="@dimen/sign_view_height"
                    android:paddingLeft="@dimen/sign_padding_left"
                    android:maxLines="1"
                    android:singleLine="true"
                    android:background="@drawable/text_shape"
                    />
                <TextView
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_marginTop="@dimen/sign_vertical_margin"
                    android:layout_marginBottom="@dimen/sign_vertical_margin"
```

```
                android:layout_marginLeft="@dimen/sign_padding_left"
                android:textSize="@dimen/sign_text_size"
                android:textColor="@color/colorGrey"
                android:text="@string/prompt_password"
                />

            <EditText
                android:id="@+id/password"
                android:inputType="textPassword"
                android:layout_width="match_parent"
                android:layout_height="@dimen/sign_view_height"
                android:paddingLeft="@dimen/sign_padding_left"
                android:maxLines="1"
                android:singleLine="true"
                android:background="@drawable/text_shape"/>

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginTop="@dimen/sign_vertical_margin"
                android:layout_marginBottom="@dimen/sign_vertical_margin"
                android:layout_marginLeft="@dimen/sign_padding_left"
                android:textSize="@dimen/sign_text_size"
                android:textColor="@color/colorGrey"
                android:text="@string/prompt_comfirm_password"/>

            <EditText
                android:id="@+id/confirmPassword"
                android:inputType="textPassword"
                android:layout_width="match_parent"
                android:layout_height="@dimen/sign_view_height"
                android:paddingLeft="@dimen/sign_padding_left"
                android:maxLines="1"
                android:singleLine="true"
                android:background="@drawable/text_shape"/>

            <Button
                android:id="@+id/sign_up_button"
                android:layout_width="match_parent"
                android:layout_height="@dimen/sign_view_height"
                android:layout_marginTop="16dp"
                android:text="@string/action_sign_up"
                android:textStyle="bold"
                android:textColor="@color/colorWhite"
                android:background="@drawable/button_shape"/>
            <Button
                android:id="@+id/sign_in_button"
                android:layout_width="match_parent"
                android:layout_marginTop="16dp"
                android:layout_height="@dimen/sign_view_height"
                android:layout_gravity="center"
                android:gravity="center"
                android:text="@string/sign_in"
                android:textSize="@dimen/sign_text_size"
                android:textColor="@color/colorGrey"
                style="?android:attr/borderlessButtonStyle"/>
        </LinearLayout>
    </LinearLayout>
</ScrollView>
</LinearLayout>
```

进度条默认不显示，当向服务器发送请求时显示，客户端接收服务端响应时隐藏，登录页同理。

（2）获取对控件的引用

打开新建的`SignUpActivity`，完成对界面控件的引用。参考代码如下：

```
public class SignUpActivity extends AppCompatActivity {

    //声明成员变量
    private AutoCompleteTextView mEmail;
    private EditText mNickname;
    private EditText mPassword;
    private EditText mConfirmPassword;
    private Button mSignUp;
    private Button mSignIn;

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

        //获取对UI组件的引用
        mEmail = (AutoCompleteTextView) findViewById(R.id.email);
        mNickname = (EditText) findViewById(R.id.nickname);
        mPassword = (EditText) findViewById(R.id.password);
        mConfirmPassword = (EditText) findViewById(R.id.confirmPassword);
        mSignUp = (Button) findViewById(R.id.sign_up_button);
        mSignIn = (Button) findViewById(R.id.sign_in_button);

    }
    ……
}
```

（3）注册点击事件

给注册界面的两个按钮设置点击事件的监听器，这一次采用和上一章不同的写法。首先在`onCreate()`方法中给两个按钮添加监听器：

```
//设置点击事件监听器
mSignUp.setOnClickListener(listener);
mSignIn.setOnClickListener(listener);
```

给这两个点击事件中传入相同的监听对象，然后我们来定义这个对象如何进行事件处理：

```
private Button mSignIn;

private View.OnClickListener listener = new View.OnClickListener() {
    @Override
    public void onClick(View v) {//传入被点击对象
        switch (v.getId()){//获取该对象的Id
            case R.id.sign_up_button:
                signUp();//尝试注册
                break;
            case R.id.sign_in_button:
                signIn();//转到登录
                break;
            default://不要忘记加上,保持编程的严谨性
                break;
        }
    }
};
```

也可以给两个按钮传入不同的监听者，分别编写不同的事件处理程序。

`signUp()`和`sigIn()`就是我们封装的代码逻辑，其中`signUp()`是注册任务的入口，`signIn()`就是跳转到登录（这里的方法名都改成了小写，小编是从C#转来写Android文档的，java里方法名都是小写开头，和C#倡导的命名规范不同，既然现在是用java开发，那就使用java的规范，小编也是反复看了好久都没有发现，当然这也不能算是一种错误，只是约定俗成的不同风格或习惯罢了）。

（上面摘抄出来的代码中的第一行是提示代码位置用的，表示下面那段代码是和声明按钮放在一起，代码位置是在一开始声明的地方，`onCreate()`方法外面。还需要注意，给大家摘出来的代码是按照步骤添加的，所以如果后面再使用同样的代码提示位置，声明一个`Handler`对象，大家要能够明白这个`Handler`是放在变量声明的位置，变量间声明的顺序也是任意的，可以把上面的`listener`放在`mSignIn`前，编译器会自己去找。）

（4）跳转到登录界面

我们已经有登录和注册两个页面了，现在就来实现两个页面之间的切换，在`signIn()`里编写下面的逻辑：

```
private void signIn() {
        //Intent:意图,要做什么事
        Intent intent=new Intent(SignUpActivity.this,SignInActivity.class);
        startActivity(intent);
    }
```

`Intent`用来描述你想要做什么，这里是要实现页面跳转，两个参数就是跳转的起点和终点。

然后我们可以把程序运行起来，看一下效果了。

记得在运行前打开应用程序清单文件`AndroidManifest.xml`，把`SignUpActivity`设置为启动项，记住任何时候一个应用中只能有一个启动项：

```
<application>
    ……
    <activity
        android:name=".ui.activity.SignUpActivity"
        android:label="@string/title_activity_sign_up"
        android:theme="@style/AppTheme.NoActionBar" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    ……
</application>
```

为了调试方便，先把注册页面设为启动项，运行后点击“转到登录”按钮观察效果。

类似地，大家也可以把登录界面的跳转逻辑自行实现了。

3.2 使用MD5保障信息安全

接下来为大家介绍在登录和注册模块会被经常使用的`MD5`算法。

`MD5`是一种散列算法，或者称为摘要算法，这是一种不可逆的运算，我们常常利用这种特性来“加密”敏感数据和进行文件校验（这里打引号是因为它不能被解密，没有相应的解密算法，当然也就不应该称其为加密算法了）。

（1）初始化SharedPreferences

为了方便讲解，我们这里使用Android为我们提供的一个用于数据存储的方便的API，也就是`SharedPreferences`。

`SharedPreferences`一般用于存储用户的偏好数据，并且可以同步到Google服务器，以允许在不同设备间同步用户设置。

在代码页完成初始化：

```
private Button mSignIn;
//声明SharedPreferences
private SharedPreferences sharedPref;
……
    onCreate(){
        ……
        //初始化,参数为文件名和保存模式
        sharedPref = getSharedPreferences("moran", MODE_PRIVATE);
    }
```

上面的`onCreate()`方法并不是一个新的方法，只是方便提示代码位置的，只写了原来的方法名，下同。

初始化的第一个参数是保存的文件名（不用加后缀），第二个参数是保存类型，这里使用`MODE_PRIVATE`表明该文件是私有的，不允许其他应用查看。

（2）向SharedPreferences保存数据

我们先把网络编程放一边，先来完成下面的逻辑。当用户点击注册按钮时，首先在本地完成数据的校验工作，逻辑代码写在`signUp()`方法中，大家可以参照上一章登录时的校验规则完成编写，在校验通过后，我们再向`SharedPreferences`提交数据：

```
if (isValid == false) {
    focusView.requestFocus();
} else {
    //新建Editor对象，往里面添加数据
    SharedPreferences.Editor editor = sharedPref.edit();
    editor.putString("email", email);
    editor.putString("nickname", nickname);
    editor.putString("password", StringUtil.getMD5(password));
    editor.commit();//把数据保存到sharedPref对象中
    Toast.makeText(this, "保存成功", Toast.LENGTH_SHORT).show();
}
```

执行完上面的代码后，我们就把数据保存到创建的`SharedPreferences`对象里了。注意：保存密码时没有直接存入密码，而是调用`StringUtil`工具类中的`getMD5()`方法，得到它的MD5值。

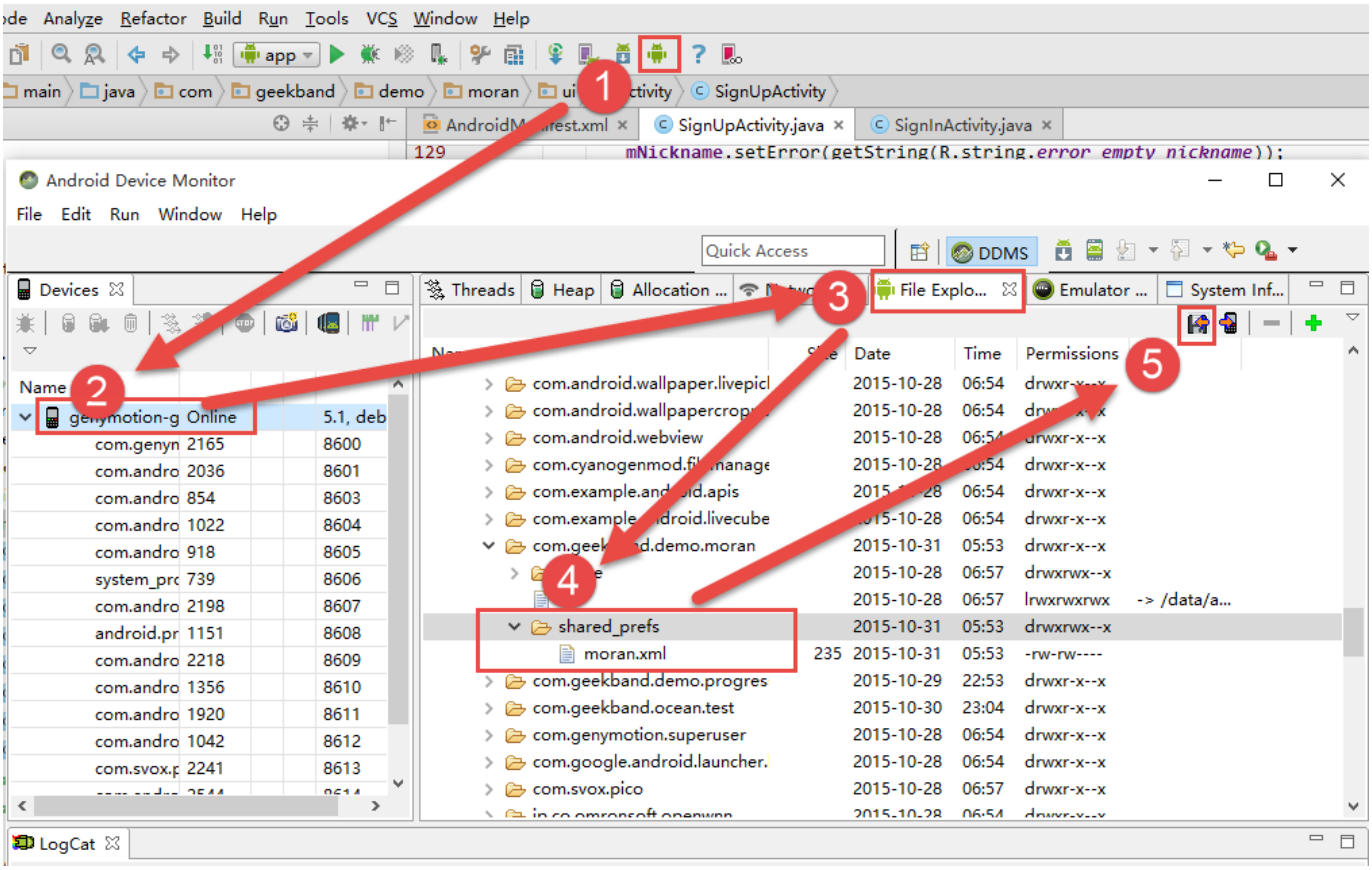
把下面的代码添加到`java/包名/util/StringUtil`文件里，以后就可以用它来获取一个字符串的`MD5`值了。

```
public class StringUtil {
    ……
    public static String getMD5(String password) {
        try {
            MessageDigest digest = MessageDigest.getInstance("md5");
            byte[] results = digest.digest(password.getBytes());
            StringBuilder stringBuilder = new StringBuilder();
            for(byte b : results){
                int number = b&0xff;
                String hex = Integer.toHexString(number);
                if(hex.length()==1){
                    stringBuilder.append("0");
                }
                stringBuilder.append(hex);
            }
            return stringBuilder.toString();
        } catch (Exception e) {
            e.printStackTrace();
            return "";
        }
    }
    ……
}
```

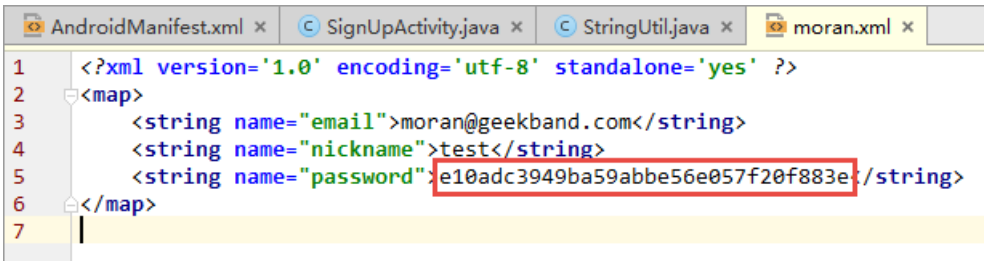
只需要拷贝两个引号之间的部分，粘贴在`StringUtil`类里边某个位置，和其他的方法同级。

（3）查看保存数据

在AS（Android Studio，下同）菜单栏，选择“Tools|Android|Android Device Monitor”，或如图点击小机器人图标，选择应用所在模拟器，在文件浏览标签页找到`data/data/包名`，在目录下就可以看到上面创建的用户偏好数据，选择xml文件点击右侧的导出图标到电脑上。



打开这个文件，我们可以看到保存的内容：



可以看到，密码字段的数据存放的是一串很难直接读懂实际意义的字符串，我们可以把敏感信息用`MD5`值的形式进行存放和传输，一定程度上保障信息安全。

当然，实际生产应用中一般不会这么简单的使用，我们还会给它“添油加醋”，比如在原来的内容基础上加一些内容再取`MD5`值。

3.3 使用多线程进行网络操作

好了，现在我们开始进行网络环境下的开发设计。

（1）完成本地校验

和登录一样，在把请求提交到服务器之前，需要先在本地完成用户输入数据的合法性校验，参考代码如下：

```
protected void onCreate(){……}
private void signIn(){……}

public void signUp() {
    //重置错误提示
    mEmail.setError(null);
    mNickname.setError(null);
    mPassword.setError(null);
    mConfirmPassword.setError(null);

    //获取注册数据
    String email = mEmail.getText().toString().trim();
    String nickname = mNickname.getText().toString().trim();
    String password = mPassword.getText().toString().trim();
    String confirmPassword = mConfirmPassword.getText().toString().trim();

    //初始化获取错误焦点的控件
    boolean isValid = true;
    View focusView = null;

    //密码验证
    if (TextUtils.isEmpty(password)) {
        mPassword.setError(getString(R.string.error_empty_password));
        focusView = mPassword;
        isValid = false;
    } else if (StringUtil.isPasswordValid(password) == false) {
        mPassword.setError(getString(R.string.error_length_password));
        focusView = mPassword;
        isValid = false;
    }

    //确认密码验证
    if (TextUtils.isEmpty(confirmPassword)) {
        mConfirmPassword.setError(getString(R.string.error_empty_password));
        focusView = mPassword;
        isValid = false;
    } else if (confirmPassword.equals(password) == false) {
        mConfirmPassword.setError(getString(R.string.error_match_password));
        focusView = mConfirmPassword;
        isValid = false;
    }

    //昵称验证
    if (TextUtils.isEmpty(nickname)) {
        mNickname.setError(getString(R.string.error_empty_nickname));
        focusView = mNickname;
        isValid = false;
    }

    //邮箱验证
    if (TextUtils.isEmpty(email)) {
        mEmail.setError(getString(R.string.error_empty_email));
        focusView = mEmail;
        isValid = false;
    } else if (StringUtil.isEmail(email) == false) {
        mEmail.setError(getString(R.string.error_pattern_email));
        focusView = mEmail;
        isValid = false;
    }

    if (isValid == false) {
        focusView.requestFocus();
    } else {
        // TODO: 提交服务器
    }
}
```

前面的`SharedPreferences`可以删掉，不需要了。完成基本的本地校验后，我们再开始网络环境下的编程。

（2）申请网络访问权限

首先确保模拟器能正常联网，模拟器不能联网的请先自行搜索解决办法。

然后，我们需要在应用清单文件`AndroidManifest.xml`里注册网络访问权限，会有两个与网络访问有关的权限：

```
<manifest>
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
</manifest>
```

这两个权限一个是允许访问网络，一个是获的网络状态信息的权限，如果不需要用到网络状态信息，可以不用加第二条权限申请。

（3）检查网络连接

这是一个非常好的习惯，在有访问网络的需求时检查网络连接是否可用。

我们可能都遇到过这种情况，比如白天在公司连上Wi-Fi，晚上回到家打开Wi-Fi设置，会发现公司的Wi-Fi名称下面显示“不在范围内”或类似信息。或者用户主动关闭了Wi-Fi和移动数据连接，导致网络连接失败，因此我们需要在网络访问时检查连接状态。

在`java/包名/util/`下新建一个`NetworkStatus`工具类，在里面填写下面的代码：

```
public class NetworkStatus {

    public static boolean isNetworkConnected(Context context) {
        if (context != null) {
            ConnectivityManager mConnectivityManager = (ConnectivityManager) context
                    .getSystemService(Context.CONNECTIVITY_SERVICE);
            NetworkInfo mNetworkInfo = mConnectivityManager.getActiveNetworkInfo();
            if (mNetworkInfo != null) {
                return mNetworkInfo.isAvailable();
            }
        }
        return false;
    }
}
```

然后，把3.3小节步骤（1）中的`TODO: 提交服务器`替换为下面的代码：

```
//检查网络连接
if (NetworkStatus.isNetworkConnected(getApplicationContext())){
    // TODO: 发送网络请求
}else {//网络不可用
    Toast.makeText(getApplicationContext(),
        getString(R.string.unavailable_network_connection),
        Toast.LENGTH_LONG).show();//为了文档能看全换行
}
```

其中`R.string.unavailable_network_connection`，在`res/values/strings.xml`里添加：

```
<resources>
    ……
    <string name="unavailable_network_connection">网络不可用，请检查Wi-Fi或移动数据连接</string>
    ……
</resources>
```

部署到模拟器上，先把模拟器的Wi-Fi关闭（如果是真机，也请关闭移动数据连接，我们以后会区分这两种网络连接状态），填写相应数据查看效果。

（4）在单独的线程上执行网络操作

由于网络操作有可能涉及到不可预知的延迟或未响应，所以我们总是应该、而且必须是把网络操作放到与主线程（UI线程）隔离的线程上，我们把上一步的`TODO: 发送网络请求`替换如下：

```
    new Thread(){
        @Override
        public void run() {
            // TODO: 提交服务器
        }
    }.start();
```

这段代码的含义是在UI线程上创建一个子线程，重写它的`run()`方法来完成相应操作，并调用它的`start()`方法启动这个线程。

（5）尝试封装表单数据

根据蓦然项目的API文档，注册模块需要以`POST`方式提交，提交的参数有`username`、`password`、`email`、`gbid`，返回的数据类型为`JSON`格式。

文档没有说明提交的格式，这里我们把提交的数据也按照`JSON`格式封装，参考代码如下：

```
new Thread(){
    @Override
    public void run() {

        try {
            //获取密码的md5值，并截取前20个字符
            String pwd=StringUtil.getMD5(password).substring(0,20);
            String gbid="GeekBand-A150010";//记得换为自己的学号
            JSONObject user = new JSONObject();//创建JSON对象
            user.put("username", nickname);//往里面添加内容
            user.put("password", pwd);
            user.put("email", email);
            user.put("gbid", gbid);
            //从全局类获取访问路径
            String url=mAppContext.getUrl(mPath);
            doPostRequest(url,user);//发送POST请求
        } catch (JSONException e) {//捕获异常
            e.printStackTrace();
        }
    }
}.start();
```

原型没有输入学号的地方，这里直接写在代码里，大家记得替换为自己的学号。

昵称提交的参数名为`username`，服务器解析时是根据这个名字来判断对应内容的，所以不要使用自定义的名称。

其中，网络访问的路径，小编放在了自定义的全局变量类（小编的位置是放在`java/包名/ApplicationContext`）中，添加代码：

```
public class ApplicationContext extends android.app.Application {

    private static String baseUrl="http://moran.chinacloudapp.cn/moran/web";

    public String getUrl(String path){
        return baseUrl+path;
    }
}
```

在`SignUpActivity`中使用该全局变量：

```
private Button mSignIn;
//使用全局变量
private ApplicationContext mAppContext;
private static final String mPath="/user/register";
……
    onCreate(){
        //获取对全局变量的引用
        mAppContext = (ApplicationContext) getApplication();
    }
```

（6）POST提交表单数据

Android包含两种访问网络的方式：`HttpURLConnection`和`HttpClient`，谷歌官方推荐在Android 2.3及以上版本中使用`HttpURLConnection`进行网络资源的请求。小编搜到一些有关这两种方式的讨论，可以查看下面的链接了解（如果打不开可以复制后面的链接到浏览器中查看）：

> “
>
> - 为什么在Android 5.1中，org.apache.http包中的类和AndroidHttpClient类均已被废弃。对开发有什么影响？(http://www.zhihu.com/question/29133606)
> - 关于安卓HTTP请求用HttpUrlConnection还是HttpClient好？(http://blog.csdn.net/huzgd/article/details/8712187)

这里就使用谷歌推荐的`HttpURLConnection`方式进行连接，完成`POST`请求的处理逻辑：

```
onCreate(){……}
signIn(){……}
signUP(){……}
private void doPostRequest(String path, JSONObject data) {
    try {
        //获得传输的实体
        byte[] entity = data.toString().getBytes("UTF-8");
        URL url = new URL(path);
        //实例化一个HTTP连接对象
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setConnectTimeout(5000);//定义超时时间
        connection.setRequestMethod("POST");//使用POST方式发送请求
        connection.setUseCaches(false);//设置不允许使用缓存
        connection.setDoOutput(true);//允许对外输出
        connection.setDoInput(true);//允许对内输入
        connection.setRequestProperty("Content-Type", "application/json");//提交格式为json
        connection.setRequestProperty("Content-Length", String.valueOf(entity.length));
        OutputStream outputStream = connection.getOutputStream();//获得输出流
        outputStream.write(entity);
        outputStream.close();

        int responseCode = connection.getResponseCode();
        if(responseCode == 200){
            // TODO: 处理成功的请求
        } else {
            // TODO: 处理失败的请求
        }

    } catch (UnsupportedEncodingException e) {//编码方面的异常
        e.printStackTrace();
    } catch (MalformedURLException e) {//路径方面的异常
        e.printStackTrace();
    } catch (ProtocolException e) {//协议方面的异常
        e.printStackTrace();
    } catch (IOException e) {//输入输出方面的异常
        e.printStackTrace();
    } catch (Exception e) {//其他方面的异常
        e.printStackTrace();
    }
}
```

`getByte()`用于将我们提交的字符串转换成字节数组，使用系统默认的字符集进行编码，这里指定使用`UTF-8`进行编码。

连接时间的单位为毫秒，最好不要超过10秒，以免被系统回收。

由于要向服务器提交提交数据，并接收服务器传回的数据，所以要设置允许输出和输入。

向服务器提交数据，对于客户端而言是输出，所以这里要使用`OutputStream`对外输出。

当执行`int responseCode = connection.getResponseCode()`这一行代码时，表单数据就会提交给服务器，服务器处理完毕，返回浏览器端响应，响应码为`200`表示服务器成功处理了请求。

网络连接过程中可能会出现各种异常，在上面摘抄的代码中最后几行就是捕获的异常种类，暂时可以不用去管它。

（7）处理响应数据

当服务器成功处理请求后，返回客户端一个输入流，我们先将输入流转换为一个字节数组，这里调用`StreamUtil`工具类中的`readInputStream()`方法：

```
if(responseCode == 200){
    InputStream inputStream=connection.getInputStream();
    byte[] is=StreamUtil.readInputStream(inputStream);
    ……
}
```

将输入流转换为字节数组的操作以后会经常使用，所以我们把处理逻辑封装起来，方便以后调用。在`java/包名/util`下新建一个名为`StreamUtil`的工具类，参考代码：

```
public class StreamUtil {
    /**
    * 从输入流中获取数据
    * @param inStream 输入流
    * @return
    * @throws Exception
    */
    public static byte[] readInputStream(InputStream inStream) throws Exception{
        ByteArrayOutputStream outStream = new ByteArrayOutputStream();
        byte[] buffer = new byte[1024];
        int len = 0;
        while( (len=inStream.read(buffer)) != -1 ){
            outStream.write(buffer, 0, len);
        }
        inStream.close();
        return outStream.toByteArray();
    }
}
```

（8）抽取需要的信息

我们现在已经得到了服务器返回的数据，还需要把数据进行解析，抽取出需要的数据，而忽略其他。

查看API文档，注册成功返回的是类似下面的数据：

```
{
"status": 1,
"data": {
"user_id": "6",
"user_name": "testProject3",
"project_id": "3"
},
"message": "Register success"
}
```

现在大家都应该清楚了，这是一串`json`格式的数据，只是里面还嵌套了一个`json`，我们简单分析一下：

最外层可以写成`{"status": 1,"data": {...},"message": "Register success"}`，这是一个`json`。

其中“data”的值又是一个`json`，即`{"user_id": "6","user_name": "testProject3","project_id": "3"}`。

对于注册模块，按照API文档暂时我们只需要`status`和`message`的就可以了，不过为了查看编码是否有问题，我们先把所有信息显示出来，接着步骤（7）第一部分的代码：

```
byte[] is = StreamUtil.readInputStream(inputStream);
String json = new String(is);

Message msg = Message.obtain();
msg.obj = json;
handler.sendMessage(msg);
```

因为现在是在子线程中操作，是无法在界面里显示信息的，所以这里使用一个`Handler`对象向主线程传递消息。

在`SignUpActivity`声明变量的地方（和其他按钮、文本框声明定义在一起，`onCreate()`之前），创建一个`Handler`对象（注意，`Handler`要使用Android系统的类库，不要用java里的，导入这个包`import android.os.Handler;`），添加处理逻辑：

```
private Button mSignIn;
……
private Handler handler;
{//这是一种将声明和初始化分隔开来的写法。
    handler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        Toast.makeText(getApplicationContext(), msg.obj.toString(), Toast.LENGTH_LONG).show();
        }
    };
}
```

`Toast`里第三个参数设置为`Toast.LENGTH_LONG`，我们让信息显示得更久一点。

现在运行程序，输入相关信息注册一下吧！如果一切正常，就会在界面上显示所有返回的数据。

这里蓦然服务器返回的信息其实是有问题的，如果输入的昵称是之前被使用过了，返回来的竟然是数据库中第一个使用这个昵称的信息。

接下来，我们就来抽取出需要的信息，紧接着步骤（7）中的代码：

```
if(responseCode == 200){//请求成功
    InputStream inputStream=connection.getInputStream();
    byte[] is=StreamUtil.readInputStream(inputStream);
    String json = new String(is);
    SONObject jsonObject = new JSONObject(json);
    int status = jsonObject.getInt("status");
    String message = jsonObject.getString("message");
    //创建消息对象
    Message msg = Message.obtain();
    msg.what=SUCCESS;
    msg.obj = message;
    handler.sendMessage(msg);
} else{//请求失败
    Message msg = Message.obtain();
    msg.what = ERROR;
    handler.sendMessage(msg);
}
```

上面的`jsonObject`用来存放抽取出的最外层的`JSON`数据，然后按照每个元素的名称解析出状态和消息两个元素。但最后小编并没有使用到`status`，因为经过小编反复尝试，发现只能返回这一种状态，API文档也只有这一种状态给出了完整示例，而其他的状态始终没有出现。

原来的逻辑是请求失败，即在`else`分支里会返回类似的数据，但如本文开篇所说，现在这个逻辑没有了或没正确执行了，所以暂时大家可以按照上面的写法，等这之后我们联系服务端的提供者了解问题。

`SUCCESS`和`ERROR`为创建的常量：

```
private static final String mPath = "/user/register";
……
private static final int SUCCESS = 1;
private static final int ERROR = 0;
```

返回消息的处理逻辑改为：

```
handler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
    switch (msg.what) {
        case SUCCESS:
            Toast.makeText(getApplicationContext(), msg.obj.toString(), Toast.LENGTH_LONG).show();
            break;
        case ERROR:
            Toast.makeText(getApplicationContext(), "注册失败", Toast.LENGTH_LONG).show();
        default:
            break;
        }
    }
};
```

这里注册失败的字符串是临时写在这里的，等以后修正蓦然服务后再调整。

好了，现在可以把部署运行起来了！

至此，注册模块也暂时告一段落了，大家现在也可以把登录模块的网络连接部分完善起来了，由于提交的密码是取的`MD5`值，所以登录时应该使用相同的逻辑，也是使用密码的`MD5`值提交登录。

实际上，即使提交时不对密码字段进行处理，服务器端也可能会做类似的操作，那我们这样做是否有必要呢？

答案当然是肯定的，而且不只是密码字段，其他信息也应该是客户端和服务器端同时做校验的，编程时应该秉持这样的原则，任何一端传递过来的信息都是不可信的，都是有可能被拦截、被篡改的，所以无论你处在哪一方，都应该实现数据的校验。

除此之外，能在本地完成的校验千万不要等到传递给服务端，让服务端去处理，而应该先在本地完成校验，比如我们在本地先完成邮箱/密码是否为空、格式是否正确、长度是否过长或过短等，完全没有必要传到服务端做处理，本地就可以完成，过滤掉这些错误之后，再让服务器处理邮箱是否已经被使用、登录密码是否正确等审核。

好，我们再把网络编程部分需要注意的内容回顾一下：

> “
>
> 1. 在程序清单里注册网络相关权限，在6.0系统上使用时还应该判断权限是否被用户回收。
> 2. 每次访问网络时先检查网络是否可用，因为设备的网络状态是随时可变的。
> 3. 在单独的线程上执行网络操作，绝不能在主线程（UI线程）上执行。
> 4. 将数据按照约定规则进行处理转换，如这里传递均为JSON格式。
> 5. 进行字符编码，建议明确指定而不是使用默认，避免出现乱码。
> 6. 选择合适的请求方式、连接时间、资源重用及释放方案。