目录

一、实验项目名称.....2
二、实验目的.....2
三、实验原理.....2
四、实验环境.....2
五、实验内容.....2
六、程序设计思路.....3
6.1 UML 需求用例图及分析说明.....3
6.2 系统总体设计.....3
6.3 系统数据库设计.....4
6.4 项目具体项目结构及内容声明.....5
6.5 应用权限设置.....7
6.6 在线新闻应用的类结构示意图.....8
七、相关功能介绍.....8
7.1 登录注册功能.....8
7.2 修改用户的个人信息.....10
7.3 在线新闻应用主页面.....11
7.4 定义应用的配置功能.....12
7.5 定义 APP 的帮助信息.....14
7.6 独创性功能.....14
八、相关技术分析.....16
8.1 Android 界面渲染技术.....16
8.2 Android 的网络连接技术.....18
8.3 Android 访问网络技术.....20
8.4 数据解析技术.....27
8.5 数据持久化处理——SharedPreferences.....28
8.6 数据持久化处理——SQLite 数据库.....32
8.7 数据持久化处理——文件处理.....34
8.8 并发处理.....35
九、实验思考与总结.....36
9.1 实验所遇问题及其解决方法.....36
9.2 实验心得与体会.....36
十、参考文献.....37

一、实验项目名称

Android 移动应用综合开发——基于 HTTP 协议的在线新闻 APP

二、实验目的

（1）学习使用 Android Studio 开发平台;

（2）学习和使用 Android 的活动组件开发移动应用;

（3）了解及掌握采用的基于 HTTP 协议实现访问网络技术、以及对网络 Json 数据的解析技术;

（4）了解和掌握 Android 界面的开发:布局、组件以及 Android 界面渲染技术（例如采用抽屉布局和 RecyclerView 组件）等高级组件的使用。

三、实验原理

（1）Android 活动的生命周期;

（2）Android 活动的活动加载方式;

（3）Android 碎片与活动的交互;

（4）Android 界面的开发:布局、组件以及 Navigation 和 RecyclerView 组件等高级组件的使用;

（5）Android 数据库的持久处理、线程的并发处理。

四、实验环境

（1）硬件: PC 微机;

（2）软件: Windows 10 以上、JDK1.8 以上、Kotlin1.3.1 以上、Android SDK、AndroidStudio、SQLite 数据库。

五、实验内容

（1）功能要求

a.要求实现登录注册功能;

b.功能设计合理和并将设计的功能完整实现;

c. 对于核心功能有独创性;

d.可以修改用户的个人信息（昵称、图片、个人简介）；

e:可以定义 APP 的帮助信息;

f:可以定义应用的配置功能，修改 APP 的字体大小、背景图片等;

g:界面设计完整，交互处理方便。

（2）技术要求

a. Android 界面渲染技术;

b. Android 的访问网络技术;

c. 数据解析技术（JSON 解析或 XML 解析;

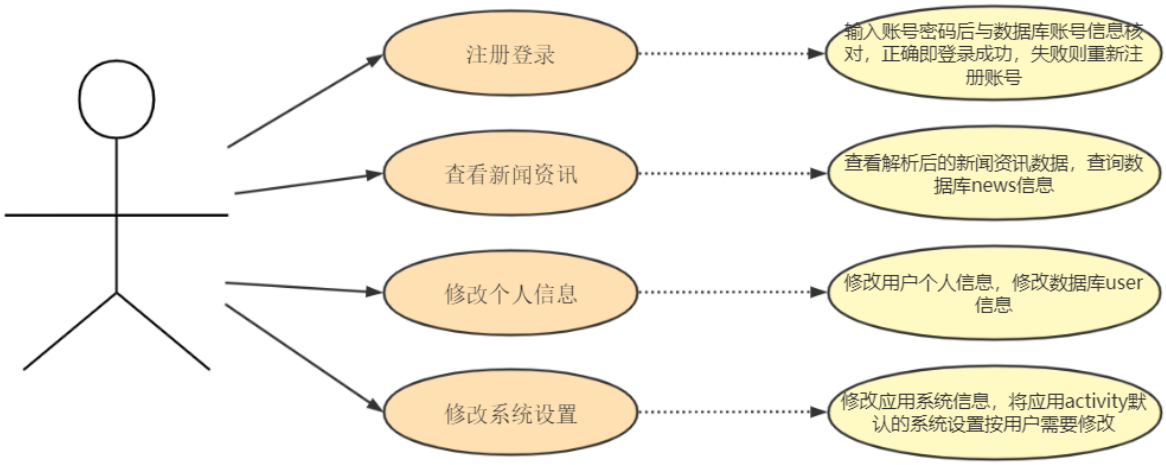
d. 数据库持久处理;

e. 文件处理;

f. 并发处理。

六、程序设计思路

6.1 UML 需求用例图及分析说明



UML 需求用例图

6.2 系统总体设计

对新闻客户端应具有的功能分析后得知，此系统主要有 4 个方面的功能，即**更新新闻、注册登录、系统设置、应用关于**。

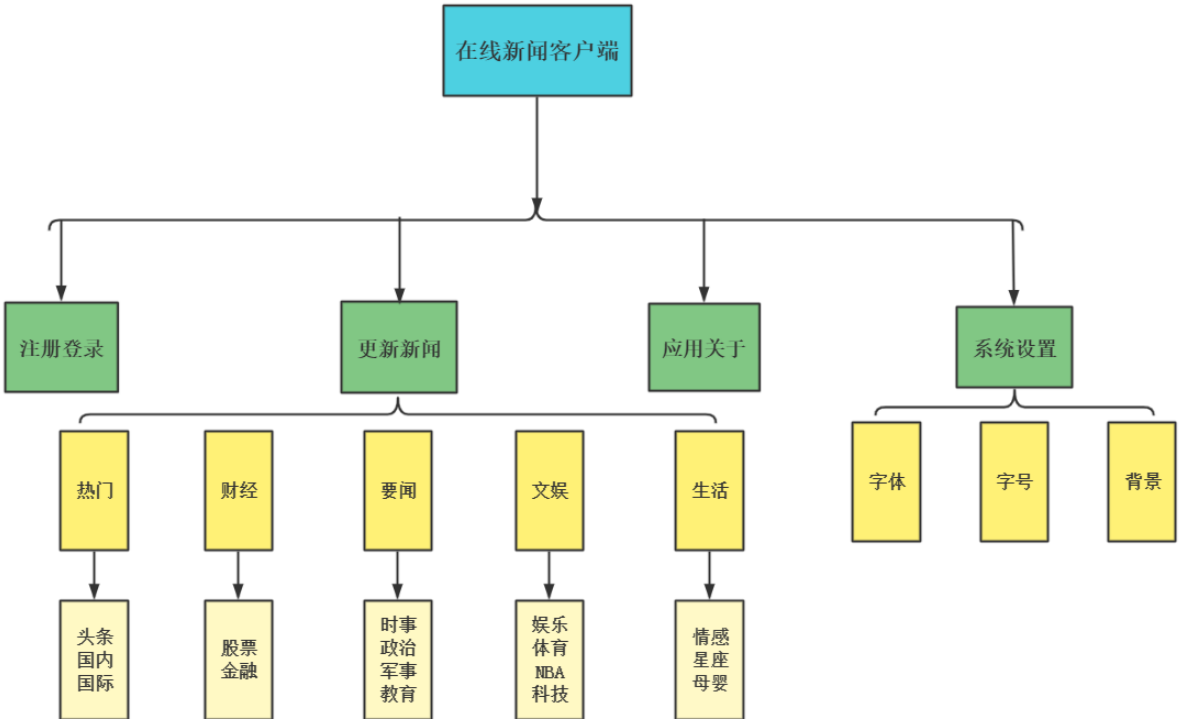
① 注册登录功能：用户使用该应用需先提供个人信息，完成个人账户信息的注册，更新数据库。

② 更新新闻功能：用户启动程序后可以进行新闻资讯的实时更新，新闻资讯分为热门、 财经、 要闻、文娱、生活等五个大模块，不同的栏目让使用者了解任意感兴趣的资讯进行阅读。

③ 系统设置功能：用户启动程序后定义应用的配置功能，可以对系统的字体、字号、背景进行选择更换。

④ 应用关于功能：新闻客户端 App 的关于功能主要是简略介绍该新闻客户端的开发者及该应用版本的作用。

系统功能架构图如下图所示：



系统功能架构图

6.3 系统数据库设计

表 1　数据项汇总列表

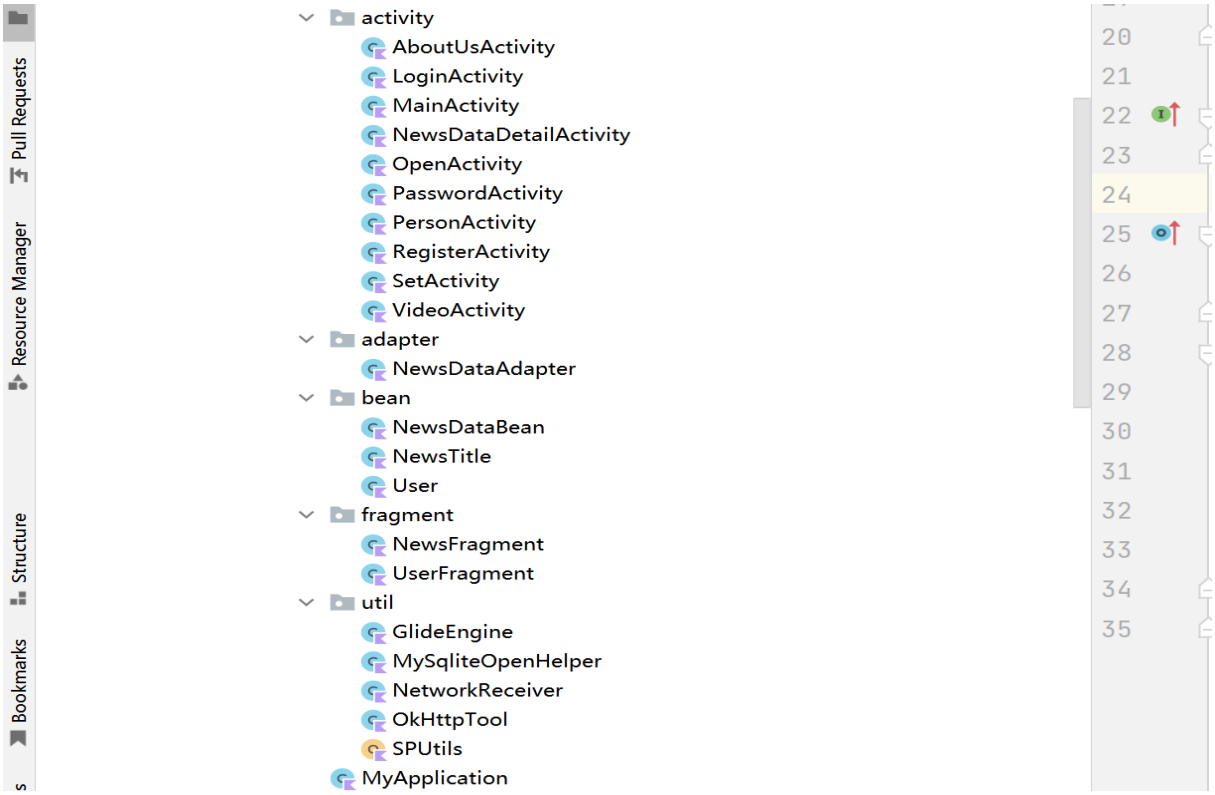
数据项编号	数据项名	数据项含义	存储结构
D0	id	用户 id 号	Integer
D1	account	用户名	Varchar(n)

D2	password	用户密码	Varchar(n)
D3	name	用户昵称	Varchar(n)
D4	sex	性别	Varchar(n)
D5	phone	用户电话	Varchar(n)
D6	address	用户简介	Varchar(n)
D7	photo	用户头像的存储位置	Varchar(n)
D8	score	用户积分	Varchar(n)
D9	typeId	新闻类型	Integer
D10	title	新闻标题	Varchar(n)
D11	img	新闻配图	Varchar(n)
D12	content	新闻内容	Varchar(n)
D13	issuer	新闻发布商	Varchar(n)
D14	date	新闻发布的日期	Varchar(n)

表 2 数据结构列表

6 数据结构编号	数据结构名称	数据结构含义	组成
S1	user	用户信息表	Account、password 、name、sex、phone、address、photo
S2	news	新闻信息表	typeId、title、img、content、issuer、date

6.4 项目具体项目结构及内容声明



系统项目结构图

以下是在线新闻应用的系统项目类及其说明表:

表 3 系统项目类及其说明表

类名	说明
MainActivity	应用程序主页面，用于显示程序主页面以及各种页面间的连接跳转
OpenActivity	欢迎页面，主要判断用户的登录状态。
LoginActivity	登录页面，用于登录用户账户
RegisterActivity	注册页面，将用户数据存入数据库中
AboutUsActivity	应用帮助页面，声明应用的帮助信息
NewsDataDetailActivity	具体新闻页面，将新闻内容平铺在应用中
PasswordActivity	重置密码页面，更新应用账户数据库

PersonActivity	个人信息页面，展示账户的个人信息
SetActivity	设置页面，修改程序系统设置
VideoActivity	视频介绍页面，内含程序运行讲解视频
NewsFragment	新闻页面,对 api 接口进行数据解析
UserFragment	用户页面，侧滑窗口展现用户个人账户信息
NewsDataBean	新闻数据解析实体类
NewsTitle	具体新闻内容解析实体类
User	利用 Serializable 方式进行序列化，将序列化的对象整体进行数据处理
GlideEngine	Glide 图片引擎类用户加载相册图片
MySqliteOpenHelper	数据库帮助类
NetworkReceiver	网络连接类
OkHttpTool	网络访问工具类用于新闻接口请求数据
SPUtils	数据持久化通用类
MyApplication	维护全局状态的基类用于保存当前 Activity 对象

6.5 应用权限设置

以下是在线新闻应用的应用权限声明表:

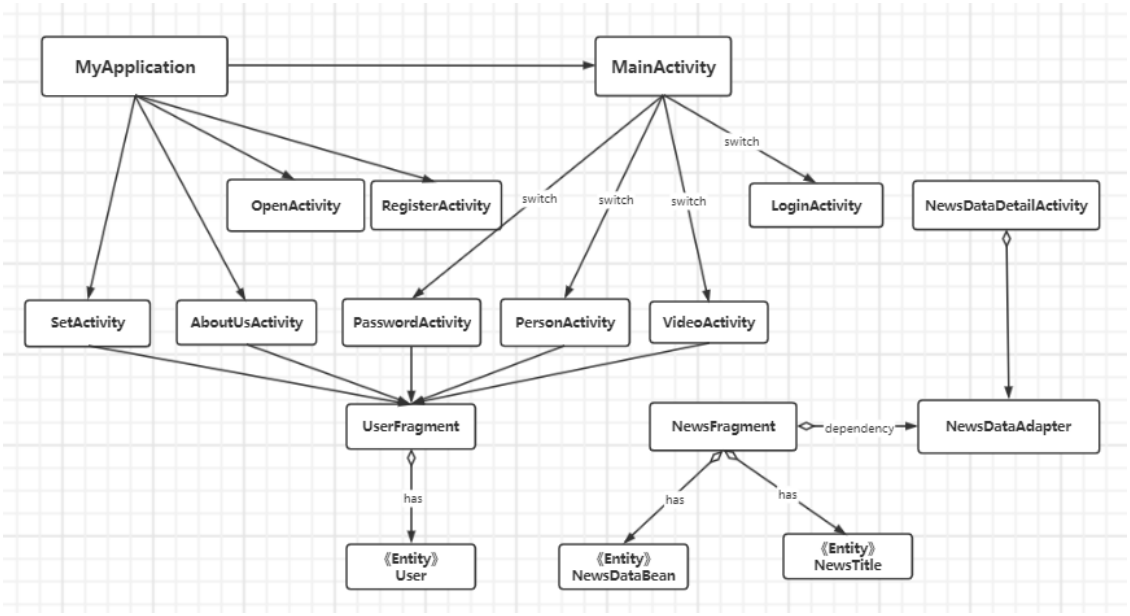
表 4 应用权限及其说明表

权限	说明
ACCESS_NETWORK_STATE	允许应用程序获取网络状态信息的权限

READ_EXTERNAL_STORAGE	设置读取外部存储的访问权限
CAMERA	允许应用程序使用照相机的权限
INTERNET	设置访问网络权限
CHANGE_WIFI_STATE	改变 WLAN 状态的开关，如果打开或关闭 Wi-Fi 必需加入
ACCESS_WIFI_STATE	获取使用 Wi-Fi 等 WLAN 无线网络

6.6 在线新闻应用的类结构示意图

以下是在线新闻应用的类关系示意图:



在线新闻应用的类关系示意图

七、相关功能介绍

以下仅展示相关功能部分图片，其他具体功能实现效果见运行视频。

7.1 登录注册功能

图 1 ： 用户登录界面
图 2 ： 当账号未注册时登录的异常报错

图 1

图 2

图 3 ： 用户注册界面

图 4 ： 当账号注册时信息填写不符合格式的异常报错

图 3

图 4

应用在上传用户头像时使用了两项运行时权限如下:

图 5 ： 上传用户头像需要获取读写外部存储权限
图 6 ： 拍摄照片需要获取照相机权限

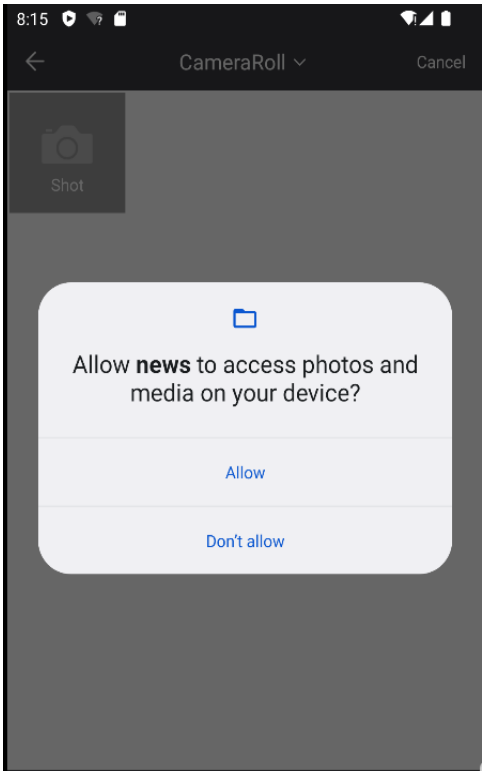


图 5

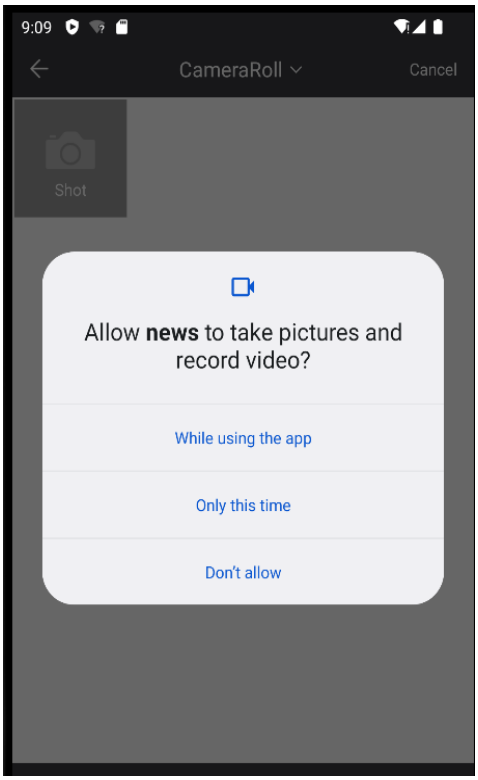
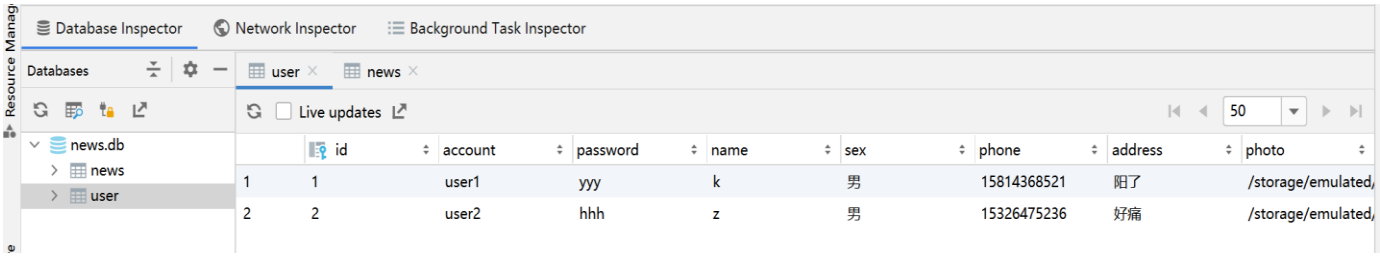


图 6

图 ： 当用户的账号注册成功，就可以在应用的 Database Inspector 中查看数据库内账号数据。



user 数据库

7.2 修改用户的个人信息

图 7 ： 原用户信息
图 8 ： 修改后用户信息

图 7

图 8

7.3 在线新闻应用主页面

图 9 ： 登录账号后应用呈现的主页面
图 10: 按解析数据的不同内容展现新闻内容

图 9

图 10

图 11 ： 点击新闻展现具体新闻内容界面
图 12: 点击新闻展现具体新闻内容界面

图 11

图 12

7.4 定义应用的配置功能

图 13、14: 应用的系统设置界面

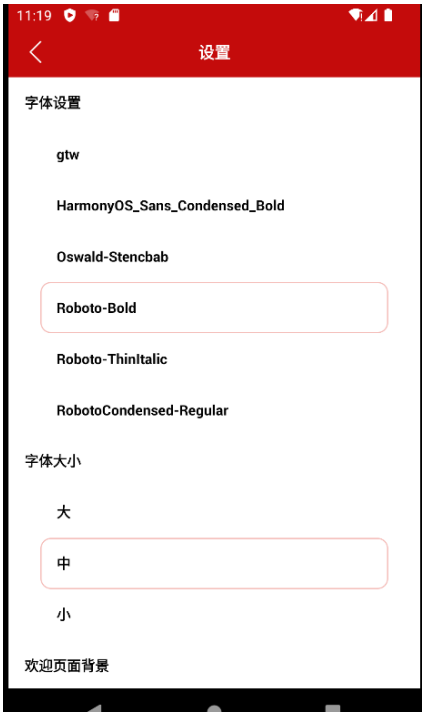


图 13

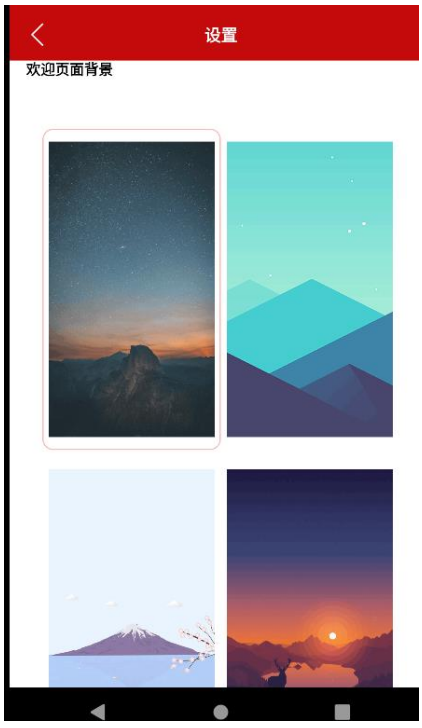


图 14

图 15: 修改字体前(原字体为 Roboto-Bold)
图 16: 修改字体后(修改后字体为 Roboto-Thinltalic)

图 15

图 16

图 17: 修改字号前(原字号大小为 1.0f)

图 18: 修改字号后(修改字号大小为 1.5f)

图 19: 修改字号前(原字号大小为 0.5f)

图 17

图 18

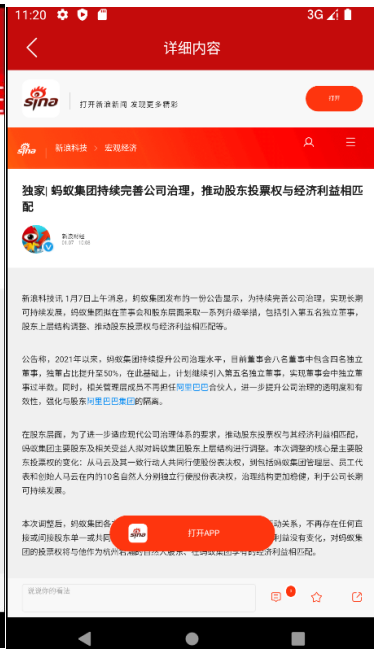


图 19

图 20: 修改背景前(原背景如下)

图 21: 修改背景后(修改背景如下)

图 20

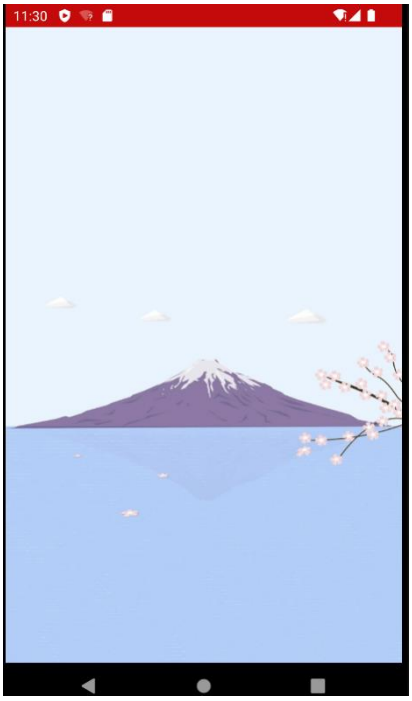


图 21

7.5 定义 APP 的帮助信息

图 22：应用的帮助界面

图 23：点击“修改”按钮修改帮助信息

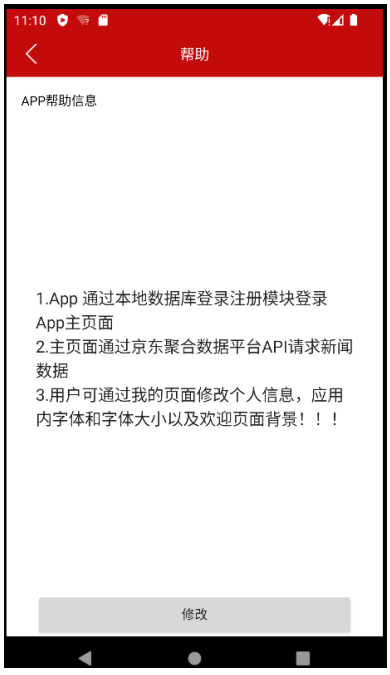


图 22

图 23

7.6 独创性功能

1.根据“学习强国”应用的启发，用户个人积分随着用户浏览时长而增加,其。UI 线程用于展现网络新闻资讯，而在应用后台利用 AsyncTask 异步执行，并结合 handler 机制用于记录当前页面的浏览时长。

图 24：账户原积分

图 25：经过一段时间浏览新闻后的账户积分

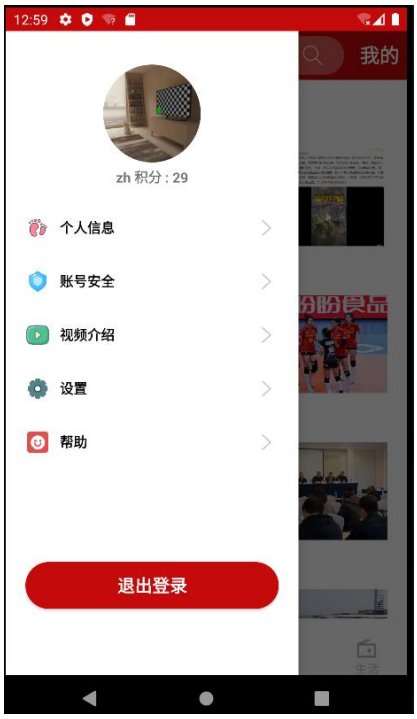


图 24

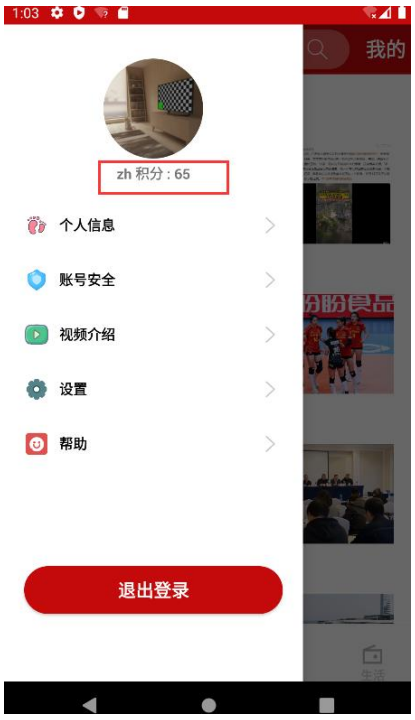


图 25

2.在用户设置个人头像时，按照正常的应用功能，不能只是立即拍照上传，而应当可以访问本地的图片库进行选择，该功能借助了 PictureSelector 工具实现快速获取本地图片/视频。具体代码见图片加载引擎代码 GlideEngine 类。

图 26：访问本地图片库

图 27：过多选择异常报错

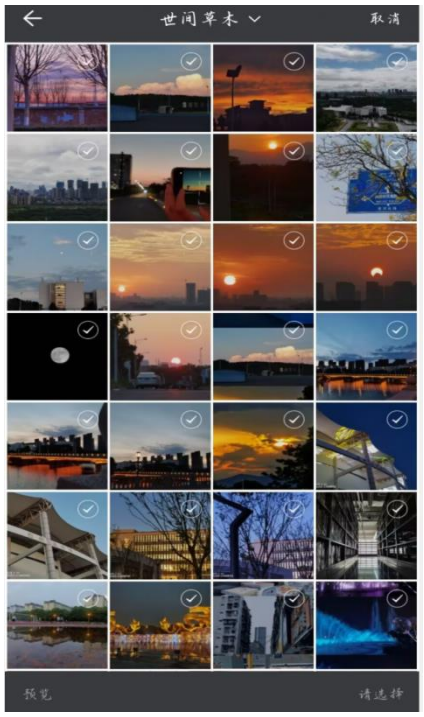


图 26　　图 27

八、相关技术分析

8.1 Android 界面渲染技术

整个应用程序的 UI 布局和主色调参照“学习强国”应用的 UI 布局，其用户页面的实现利用了 drawerlayout 抽屉布局进行呈现，而新闻资讯的排布使用了 RecyclerView 组件实现。其呈现效果如下图:

图 28: 抽屉布局展现用户个人信息。
图 29: 使用 RecyclerView 组件排列展现新闻内容。

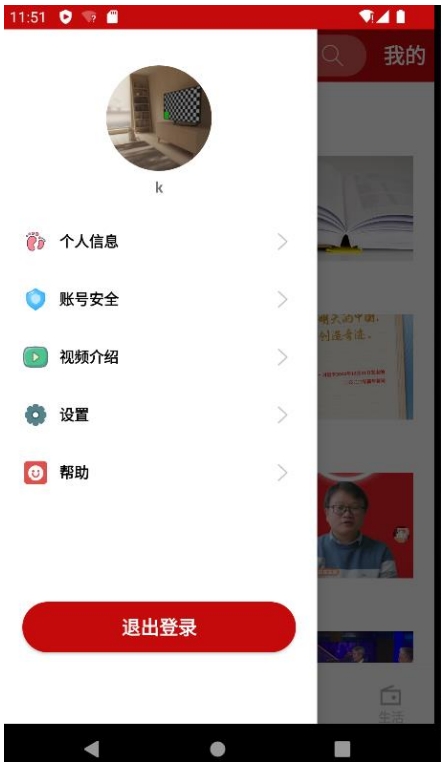


图 28　　图 29

相关技术代码及分析如下:

drawerlayout 抽屉布局只需要在对应布局 main_xml 文件中添加

```
<androidx.drawerlayout.widget.DrawerLayout>
</androidx.drawerlayout.widget.DrawerLayout>
```

在 build.gradle 中添加相关依赖:

```
implementation 'androidx.recyclerview:recyclerview:1.2.1'
```

在 Newsfragment 中内嵌 RecyclerView 组件，其初始化 UI 布局代码如下:

```
//初始化页面
private fun initView() {
    loadTitle()
```

```
    //1.1、设置线性布局
    val layoutManager = LinearLayoutManager(myActivity)
    //1.2、设置为垂直排列，用 setOrientation 方法设置(默认为垂直布局)
    layoutManager.orientation = LinearLayoutManager.VERTICAL
    //1.3、设置 recyclerView 的布局管理器
    rvNewsList.layoutManager = layoutManager
    //2.1、初始化适配器
    mNewsAdapter = NewsDataAdapter()
    //2.2、设置 recyclerView 的适配器
    rvNewsList.adapter = mNewsAdapter
    tabTitle.addOnTabSelectedListener(object : OnTabSelectedListener {
        override fun onTabSelected(tab: TabLayout.Tab) {
            type = titles[tab.position]
            loadData()
        }
        override fun onTabUnselected(tab: TabLayout.Tab) {}
        override fun onTabReselected(tab: TabLayout.Tab) {}
    })
}
```

对应的 ite_rv_new_data_list.xml 的布局文件具体代码如下:

```
    <?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="wrap_content"
    android:padding="20dp"
    android:background="@color/colorWhite"
    android:layout_marginTop="10dp">
    <LinearLayout
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:layout_marginRight="20dp"
        android:orientation="vertical">
        <TextView
            android:id="@+id/title"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
```

```
            android:text="标题"
            android:textSize="16dp"
            android:textColor="@color/colorBlack"
            android:textStyle="bold"/>
        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:orientation="horizontal">
            <TextView
                android:id="@+id/author_name"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginRight="10dp"
                android:text="新京报"/>
            <TextView
                android:id="@+id/date"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="2021 年 3 月 28 日 12:32:37"/>
        </LinearLayout>
    </LinearLayout>
    <ImageView
        android:id="@+id/thumbnail_pic_s"
        android:layout_width="130dp"
        android:layout_height="100dp"
        android:scaleType="centerCrop"/>
</LinearLayout>
```

8.2 Android 的网络连接技术

图 30: 使用移动网络登录时的提示信息。

图 31: 使用 WIFI 网络登录时的提示信息。

图 30

图 31

相关技术代码及分析如下:

在做 Android app 的时候，网络的状态可能会经常的变化，当用户当前没有网络的时候，我们要保证程序不崩溃，而在当用户使用移动网络时，则需要给予用户提醒。

查阅官方文档，想要获取网络状态的话，我们需要通过 ConnectivityManager 类来实现，该类负责与 NetworkManagementService 系统服务进行交互来获取网络相关的信息。

NetworkInfo 是一个描述网络连接状态的接口，我们主要通过这个类来获取网络的状态。

```
val connectivityManager: ConnectivityManager =
context.getSystemService(ConnectivityManager::class.java) as ConnectivityManager
val networkInfo = connectivityManager.activeNetworkInfo
```

从官方文档我们可以知道，如果要想监听到网络的变化的，那么需要我们注册一个广播才行,以下是网络连接的广播类:

```
class NetworkReceiver : BroadcastReceiver() {
override fun onReceive(context: Context, intent: Intent) {
    val action = intent.action
    //判断有无网络连接
    if (action == ConnectivityManager.CONNECTIVITY_ACTION) {
        val connectivityManager = context.getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager
        val networkInfo = connectivityManager.activeNetworkInfo
        //说明当前有网络
        if (networkInfo != null && networkInfo.isAvailable) {
```

```
                val type = networkInfo.type
                when (type) {
                    ConnectivityManager.TYPE_MOBILE -> Toast.makeText(context, "当前移动网络正常", Toast.LENGTH_SHORT).show()
                    ConnectivityManager.TYPE_WIFI -> Toast.makeText(context, "当前 WIFI 网络正常", Toast.LENGTH_SHORT).show()
                    ConnectivityManager.TYPE_ETHERNET -> Toast.makeText(context, "当前以太网网络正常", Toast.LENGTH_SHORT).show()
                }
            } else {
                //说明当前没有网络
                Toast.makeText(context, "当前网络异常", Toast.LENGTH_SHORT).show()
            }
        }
    }
}
```

在 MaiActicity 中产生 NetworkReceiver()的实例:

```
    private fun initNetworkReceiver() {
    mNetReceiver = NetworkReceiver()
    val filter = IntentFilter()
    filter.addAction(ConnectivityManager.CONNECTIVITY_ACTION)
    registerReceiver(mNetReceiver, filter)
}
```

在销毁 Activity 的同时注销网络监听广播:

```
    override fun onDestroy() {
    super.onDestroy()
    unregisterReceiver(mNetReceiver)
}
```

8.3 Android 访问网络技术

在实现在线新闻 app 时，访问网络技术是其应用主要功能的前提，如何在手机端使用 HTTP 和服务器进行网络交互，并对服务器返回的复杂数据进行解析是应用实现的主要难题。

OKHttp 是一个当前主流的网络请求的开源框架，由 Square 公司开发，用于替代 HttpUrlConnection 和 Apache HttpClient。Retrofit 库的网络请求的工作本质上就是由 OkHttp 完成的，Retrofit 仅负责网络请求接口的封装。

相关技术代码及分析如下:

首先在 build.gradle 添加相关依赖:

```
implementation 'com.squareup.okhttp3:okhttp:5.0.0-alpha.2'
implementation 'com.squareup.okhttp3:logging-interceptor:4.4.0'
```

在应用配置文件中设置网络权限:

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

构建 OkHttpTool 工具类，便于实用 Okhttp3,代码如下:

```
class OkHttpTool {
    //获取图片  返回
    fun getImage(url: String?) {
        var response: Bitmap
        OkHttpUtils
                .get() //
                .url(url) //
                .tag(this) //
                .build() //
                .connTimeOut(20000) //链接超时
                .readTimeOut(20000) //读取超时
                .writeTimeOut(20000) //写入超时
                .execute(object : BitmapCallback() {
                    override fun onError(request: Request, e: Exception) {}
                    override fun onResponse(response: Bitmap) {}
                })
    }
    //回调接口
    interface ResponseCallback {
        fun onResponse(isSuccess: Boolean, responseCode: Int, response: String?, exception:
Exception?)
    }
    companion object {
        //日志标志
        private const val TAG = "OkHttpTool"
        //OkHttpClient 类
        private var myOkHttpClient: OkHttpClient? = null
        init {
            //cookie 处理--让服务端记住 app
            //这里是设置 cookie 的,但是并没有做持久化处理;只是把 cookie 保存在内存中
            val cookieJar: CookieJar = object : CookieJar {
```

```
                private val cookieStore = HashMap<String, List<Cookie>>()

                //保存 cookie
                override fun saveFromResponse(url: HttpUrl, cookies: List<Cookie>) {
                    cookieStore[url.host] = cookies
                }
                //获取 cookie
                override fun loadForRequest(url: HttpUrl): List<Cookie> {
                    val cookies = cookieStore[url.host]
                    return cookies ?: ArrayList()
                }
            }
            //创建 OkHttpClient
            myOkHttpClient = OkHttpClient.Builder()
                    .connectTimeout(10, TimeUnit.SECONDS) //连接超时
                    .writeTimeout(20, TimeUnit.SECONDS) //写入超时
                    .readTimeout(20, TimeUnit.SECONDS) //读取超时
                    .cookieJar(cookieJar)
                    .build()
        }
        //================对外方法====================
        // Get 请求
        //参数: 请求地址、请求参数、请求回调
        fun httpGet(url: String, parameters: MutableMap<String, Any>, responseCallback: Re
sponseCallback) {
            val request = createGetRequest(url, parameters)
            doRequest(request, responseCallback)
        }
        //POST 请求
        //参数: 请求地址、请求参数、请求回调
        fun httpPost(url: String, parameters: Map<String, Any>?, responseCallback: Response
Callback) {
            val request = createPostRequest(url, parameters)
            doRequest(request, responseCallback)
        }
        // POST 请求 JSON 格式参数
        //参数: 请求地址、请求参数、请求回调
        fun httpPostJson(url: String, json: String, responseCallback: ResponseCallback) {
```

```
            val request = createPostRequestJson(url, json)
            doRequest(request, responseCallback)
        }
        //构建 Get 请求
        private fun createGetRequest(url: String, parameters: Map<String, Any>): okhttp3.Request {
            val urlBuilder = StringBuilder()
            urlBuilder.append(url)
            if (url.indexOf('?') <= -1) {
                //未拼接参数
                urlBuilder.append("?")
            }
            for ((key, value) in parameters) {
                urlBuilder.append("&")
                urlBuilder.append(key)
                urlBuilder.append("=")
                urlBuilder.append(value.toString())
            }
            return baseRequest.url(urlBuilder.toString()).build()
        }
        //构建 POST 请求
        private fun createPostRequest(url: String, parameters: Map<String, Any>?): okhttp3.Request {
            val builder = FormBody.Builder(Charset.forName("UTF-8"))
            if (parameters != null) {
                for ((key, value) in parameters) {
                    builder.add(key, value.toString())
                }
            }
            val formBody: FormBody = builder.build()
            return baseRequest.url(url).post(formBody).build()
        }
        //构建 POST JSON 格式参数请求
        private fun createPostRequestJson(url: String, json: String): okhttp3.Request {
            val mediaType: MediaType? = "application/json;charset=utf-8".toMediaTypeOrNull()
            val body: RequestBody = create( mediaType,json)
            return baseRequest.url(url).post(body).build()
```

```
        }
        //实际进行请求的方法
        private fun doRequest(request: okhttp3.Request, responseCallback: ResponseCallback)
{
            //使用 okhttp3 的异步请求
            myOkHttpClient!!.newCall(request).enqueue(object : Callback {
                //失败回调
                override fun onFailure(call: Call, e: IOException) {
                    //回调
                    responseCallback.onResponse(false, -1, null, e)
                    //用于输出错误调试信息
                    if (e.message != null) {
                        Log.e(TAG, e.message!!)
                    }
                }
                //成功回调
                @Throws(IOException::class)
                override fun onResponse(call: Call, response: Response) {
                    val responseCode = response.code //获取响应码
                    val responseBody = response.body //获取 ResponseBody
                    if (response.isSuccessful && responseBody != null) {
                        val strResponse = responseBody.string()
                        //回调
                        responseCallback.onResponse(true, responseCode, strResponse, null)
                    } else {
                        //回调
                        responseCallback.onResponse(false, responseCode, null, null)
                    }
                }
            })
        }
        //获取请求 指定 client 为 Android
        private val baseRequest: okhttp3.Request.Builder
            private get() {
                val builder = okhttp3.Request.Builder()
                builder.addHeader("client", "Android")
                return builder
            }
```

```
    }
```

在 NewsFragment 中实现访问网络数据，在 loadTitle()中调用 OKhttpTool 工具类的 httpGet()函数，并对回调响应接口 ResponseCallback 中的 onResponse 函数进行重写,代码如下:

```
httpGet(url, map, object : ResponseCallback {
    override fun onResponse(isSuccess: Boolean, responseCode: Int, response: String?, excepti
on: Exception?) {
        myActivity.runOnUiThread {
            if (isSuccess && responseCode == 200) {
                val newsTitle = gson.fromJson(response, NewsTitle::class.java)
                if ("10000" == newsTitle.code) {
                    Log.e("onResponse", "onResponse titles size is " + newsTitle.result!!.re
sult!!.size)
                    when (dataType) {
                        "0" -> {
                            titles.add(newsTitle.result!!.result?.get(0) ?: "") //头条
                            titles.add(newsTitle.result!!.result?.get(3) ?: "") //国际
                            titles.add(newsTitle.result!!.result?.get(2) ?: "") //国内
                        }
                        "1" -> {
                            titles.add(newsTitle.result!!.result?.get(12) ?:"") //股票
                            titles.add(newsTitle.result!!.result?.get(5) ?: "") //财经
                        }
                        "2" -> {
                            titles.add(newsTitle.result!!.result?.get(1) ?: "") //新闻
                            titles.add(newsTitle.result!!.result?.get(4) ?: "") //政治
                            titles.add(newsTitle.result!!.result?.get(8) ?: "") //军事
                            titles.add(newsTitle.result!!.result?.get(9) ?: "") //教育
                        }
                        "3" -> {
                            titles.add(newsTitle!!.result!!.result?.get(6) ?: "") //体育
                            titles.add(newsTitle.result!!.result?.get(7) ?: "") //娱乐
                            titles.add(newsTitle.result!!.result?.get(10) ?: "") //科技
                            titles.add(newsTitle.result!!.result?.get(11) ?: "") //NBA
                        }
                        "4" -> {
                            titles.add(newsTitle.result!!.result?.get(13) ?: "") //星座
                            titles.add(newsTitle.result!!.result?.get(14) ?: "") //女性
                            titles.add(newsTitle.result!!.result?.get(15) ?: "") //育儿
```

```
                        }
                    }
                    // titles.addAll(newsTitle.getResult().getResult());
                    tabTitle.tabMode = TabLayout.MODE_SCROLLABLE
                    //为 TabLayout 添加 tab 名称
                    for (i in titles.indices) {
                        when (titles.get(i)) {
                            "财经" -> tabTitle.addTab(tabTitle.newTab().setText("金融"))
                            "新闻" -> tabTitle.addTab(tabTitle.newTab().setText("时事"))
                            "女性" -> tabTitle.addTab(tabTitle.newTab().setText("情感"))
                            "育儿" -> tabTitle.addTab(tabTitle.newTab().setText("母婴"))
                            else -> tabTitle.addTab(tabTitle.newTab().setText(titles.get(i)))
                        }
                    }
                    type = titles.get(0)
                    loadData()
                }
            }
        }
    }})
```

在上述代码的 loadData()中调用 OKhttpTool 工具类的 httpGet( ) 函数，并对回调响应接口 ResponseCallback 中的 onResponse 函数进行重写,代码如下:

```
    httpGet(url, map, object : ResponseCallback {
    override fun onResponse(isSuccess: Boolean, responseCode: Int, response: String?, excepti
on: Exception?) {
        myActivity.runOnUiThread {
            if (isSuccess && responseCode == 200) {
                try {
                    val jsonObject = JSONObject(response)
                    val result = jsonObject.getString("result")
                    val jsonResult = JSONObject(result)
                    val result2 = jsonResult.getString("result")
                    val jsonResult2 = JSONObject(result2)
                    val list = jsonResult2.getString("list")
                    val type = object : TypeToken<List<NewsDataBean?>?>() {}.type //列
表信息
```

```
                    newsDataBeanList = gson.fromJson(list, type)
                    if (newsDataBeanList.size > 0) {
                        mNewsAdapter.addItem(newsDataBeanList)
                    }
                } catch (e: JSONException) {
                    e.printStackTrace()
                }
            }
        }
    }
})
```

8.4 数据解析技术

在成功访问网络资源后，需要对获得的 Json 数据进行提取相关信息。对于复杂的 Json 数据，手动编写 Json 数据的解析函数会较为复杂，且针对不同的 api 接口的 json 的数据需要重新定义解析函数，因此我利用了 Gson 解析器来实现 json 数据的解析。

相关技术代码及分析如下:

首先在 build.gradle 添加相关依赖:

```
implementation 'com.google.code.gson:gson:2.10'
```

查看 api 接口的 Json 数据格式:


```
{
  "code": "10000",
  "charge": false,
  "msg": "查询成功",
  "result": {
    "status": 0,
    "msg": "ok",
    "result": {
      "channel": "头条",
      "num": 40,
      "list": [
        {
          "title": "焦点访谈丨稳订单 拓市场 增信心",
          "time": "2023-01-06",
          "src": "央视网",
          "category": "news",
          "pic": "https://n.sinaimg.cn/spider20230106/400/w1280h720/20230106/7a14-69154537cd8629d8b465a1c13faa720e.png",
          "url": "https://news.sina.cn/2023-01-07/detail-imxzhkzz5690322.d.html?vt=4&pos=108",
          "weburl": "https://news.sina.com.cn/c/xl/2023-01-07/doc-imxzhkzz5690322.shtml",
          "content": "<p class=\"art_p\">去年12月以来，全国各地的抢单团纷纷出海。浙江、江苏、广东、四川、福建等地，一架架包机满载着中国企业家们飞赴世界各地，这一个个抢单团都是各
地政府根据当地产业发展优势“量身打造”，行程“又多又实”。面对被疫情影响了三年的市场，企业最迫切的就是与客户见面互动，为当下和未来的合作创造更多可能。
</p>\n\r\n\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t<figure class=\"art_img_mini j_p_gallery\">\r\n\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t<img class=\"art_img_mini_img j_fullppt_cover\" w=\"1280\" h=\"720\"
src='https://n.sinaimg.cn/spider20230106/400/w1280h720/20230106/7a14-69154537cd8629d8b465a1c13faa720e.png' alt=\"\" />\r\n\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t<h2 class=\"art_img_tit\">
</h2>\r\n\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t</figure>\r\n\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\n<p class=\"art_p\">去年12月初的几天，随着疫情防控措施优化调整，嘉兴商务局外贸处的陆建良忙得不可开交。</p>\n<p
class=\"art_p\">纺织业是浙江嘉兴的传统强势产业。对嘉兴而言，日本一直是重要的贸易合作伙伴。去年12月初，日本AFF纺织成衣服装服饰展览会即将开幕，这是疫情防控政策优化调整后的一次重要
商机。陆建良一头忙着确定赴日参展企业名单，一头忙着协调政府部门，促成这次包机出行。</p>\n\r\n\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t<figure class=\"art_img_mini
j_p_gallery\">\r\n\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t<img class=\"art_img_mini_img j_fullppt_cover\" w=\"1280\" h=\"720\"
src='https://n.sinaimg.cn/spider20230106/400/w1280h720/20230106/54fa-ead59c6ad15f589e25c5e4b0e7f575b9.png' alt=\"\" />\r\n\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t<h2 class=\"art_img_tit\">
</h2>\r\n\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t</figure>\r\n\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\n<p class=\"art_p\">为了多争取一点时间，陆建良牵头起草了十几份报告，向各个上级部门申请，多方沟通，原本5天的签证破例
延长到了7天。</p>\n<p class=\"art_p\">防疫、签证、航班、展位，短短几天所有工作都要到位，各部门协作配合，都打破了常规，为企业出行打开绿色通道。
</p>\n\r\n\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t<figure class=\"art_img_mini j_p_gallery\">\r\n\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t<img class=\"art_img_mini_img j_fullppt_cover\" w=\"1280\" h=\"720\"
src='https://n.sinaimg.cn/spider20230106/400/w1280h720/20230106/28df-b8d0ebf1622984c898b43fdeb4b6933f.png' alt=\"\" />\r\n\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t<h2 class=\"art_img_tit\">
</h2>\r\n\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t</figure>\r\n\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\n<p class=\"art_p\">孔志明做了30年外贸，他的3层外贸工厂，眼下只有2楼的半幅生产线还在开工。一直以来，对日出口纺织品服
装占我国全球纺织出口总额的10%以上，我国在日本纺织品服装进口市场上也占据着75%以上的份额。可是，新冠疫情暴发三年，对全球产业链供应链造成了巨大冲击。老客户已经三年没有见面，一些订
单流向了东南亚。</p>\n<p class=\"art_p\">在嘉兴，孔志明这样的困境，不是个例，有80%的企业在手订单，预计排产时间已经降到了三个月以内。这一次政府部门包机出行，有50多家企业近百人报
名参加。</p>\n\r\n\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t<figure class=\"art_img_mini j_p_gallery\">\r\n\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t<img class=\"art_img_mini_img j_fullppt_cover\" w=\"1280\" h=\"720\"
src='https://n.sinaimg.cn/spider20230106/400/w1280h720/20230106/cdca-0c08ba5359ccccb6d9faa4c40ee1319d.png' alt=\"\" />\r\n\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t"
        }
```


针对响应数据设置实体类，以下是 NewsTitle 的数据格式:

```
class NewsTitle {
var code: String? = null
var charge = false
var msg: String? = null
var result: Result? = null
```

```
    inner class Result {
        var status: String? = null
        var msg: String? = null
        var result: List<String>? = null
    }
}
```

以下是 NewsDataBean 的数据格式:

```
    class NewsDataBean {
    var title: String? = null
    var time: String? = null
    var src: String? = null
    var category: String? = null
    var pic: String? = null
    var content: String? = null
    var url: String? = null
    var weburl: String? = null
}
```

为了避免使用 Gson 时遇到 locale 影响 Date 格式的问题，使用 GsonBuilder 来创建 Gson 对象，在创建过程中调用 GsonBuilder.setDateFormat(String)指定一个固定的格式。

```
    private val gson = GsonBuilder().setDateFormat("yyyy-MM-dd HH:mm:ss").create()
```

对 Json 数据进行解析，代码如下:

利用 Gson.fromJson 函数解析出 newsTitle:

```
    val newsTitle = gson.fromJson(response, NewsTitle::class.java)
```

由于需要解析出 newsDataBeanList，于是需要通过 TypeToken 来实现转换

```
    val type = object : TypeToken<List<NewsDataBean?>?>() {}.type //列表信息
newsDataBeanList = gson.fromJson(list, type)
if (newsDataBeanList.size > 0) {
    mNewsAdapter.addItem(newsDataBeanList)
}
```

8.5 数据持久化处理——SharedPreferences

SharedPreferences 是 Android 中常用的数据存储方式，SharedPreferences 采用 key-value (键值对)形式，主要用于轻量级的数据存储，尤其适合保存应用的配置参数，因此我此处用来保存应用的系统配置数据。

相关技术代码及分析如下:

由于在平时写代码的过程中，可能会用到使用 SharedPreferences 来保存数据，用一次写一次保存的过程，会导致代码重复率太多，于是对 SharedPreference 的使用做了封装，对外公布出 put，get，remove，clear 等方法。

在 setActivity 中对系统的字体、字号、背景等系统设置进行修改时调用了对外公布出 put 方法,代码如下:

```
    private fun changeBg(type: Int) {
    put(this, "bg", type)
    reShow()
    Toast.makeText(this, "该功能需重启 APP 后生效!!!", Toast.LENGTH_SHORT).show()
}

private fun changeFontSize(tt: Int) {
    put(this, "tt", tt)
    reShow()
    Toast.makeText(this, "该功能需重启 APP 后生效!!!", Toast.LENGTH_SHORT).show()
}

private fun changeFont(fontS: String) {
    put(this, "fonts", "fonts/$fontS.ttf")
    reShow()
    Toast.makeText(this, "该功能需重启 APP 后生效!!!", Toast.LENGTH_SHORT).show()
}
```

SPUtils 里面所有的 commit 操作使用了 SharedPreferencesCompat.apply 进行了替代，目的是尽可能的使用 apply 代替 commit，在找不到的情况下才使用 commit，因为 commit 方法是同步的，并且我们很多时候的 commit 操作都是 UI 线程中，尽可能异步实现；所以我们使用 apply 进行替代，利用 apply 异步的进行写入。

以下是数据持久化的工具类:

```
object SPUtils {
    const val IF_FIRST = "is_first" //是否第一次登陆
    const val IS_ADMIN = "is_admin" //是否是管理员
    const val USER_ID = "USER_ID" //账号
    //保存在手机里面的文件名
    private const val FILE_NAME = "share_data"
    // 保存数据的方法，我们需要拿到保存数据的具体类型，然后根据类型调用不同的保存方法
```

```
    fun put(context: Context, key: String?, `object`: Any) {
        val sp = context.getSharedPreferences(FILE_NAME,
            Context.MODE_PRIVATE)
        val editor = sp.edit()
        if (`object` is String) {
            editor.putString(key, `object`)
        } else if (`object` is Int) {
            editor.putInt(key, `object`)
        } else if (`object` is Boolean) {
            editor.putBoolean(key, `object`)
        } else if (`object` is Float) {
            editor.putFloat(key, `object`)
        } else if (`object` is Long) {
            editor.putLong(key, `object`)
        } else {
            editor.putString(key, `object`.toString())
        }
        SharedPreferencesCompat.apply(editor)
    }
    //得到保存数据的方法，我们根据默认值得到保存的数据的具体类型，然后调用相对于的
方法获取值
    operator fun get(context: Context, key: String?, defaultObject: Any?): Any? {
        return try {
            val sp = context.getSharedPreferences(FILE_NAME,
                Context.MODE_PRIVATE)
            if (defaultObject is String) {
                return sp.getString(key, defaultObject as String?)
            } else if (defaultObject is Int) {
                return sp.getInt(key, (defaultObject as Int?)!!)
            } else if (defaultObject is Boolean) {
                return sp.getBoolean(key, (defaultObject as Boolean?)!!)
            } else if (defaultObject is Float) {
                return sp.getFloat(key, (defaultObject as Float?)!!)
            } else if (defaultObject is Long) {
                return sp.getLong(key, (defaultObject as Long?)!!)
            }
            null
        } catch (e: Exception) {
```

```
            defaultObject
        }
    }
    //移除某个 key 值已经对应的值
    fun remove(context: Context, key: String?) {
        val sp = context.getSharedPreferences(FILE_NAME,
            Context.MODE_PRIVATE)
        val editor = sp.edit()
        editor.remove(key)
        SharedPreferencesCompat.apply(editor)
    }
    // 清除所有数据
    fun clear(context: Context) {
        val sp = context.getSharedPreferences(FILE_NAME,
            Context.MODE_PRIVATE)
        val editor = sp.edit()
        editor.clear()
        SharedPreferencesCompat.apply(editor)
    }
    //查询某个 key 是否已经存在
    fun contains(context: Context, key: String?): Boolean {
        val sp = context.getSharedPreferences(FILE_NAME,
            Context.MODE_PRIVATE)
        return sp.contains(key)
    }
    // 返回所有的键值对
    fun getAll(context: Context): Map<String, *> {
        val sp = context.getSharedPreferences(FILE_NAME,
            Context.MODE_PRIVATE)
        return sp.all
    }
    //创建一个解决 SharedPreferencesCompat.apply 方法的一个兼容类
    private object SharedPreferencesCompat {
        private val sApplyMethod = findApplyMethod()
        // 反射查找 apply 的方法
        private fun findApplyMethod(): Method? {
            try {
                val clz: Class<*> = SharedPreferences.Editor::class.java
```

```
                return clz.getMethod("apply")
            } catch (e: NoSuchMethodException) {
            }
            return null
        }
        //如果找到则使用 apply 执行，否则使用 commit
        fun apply(editor: SharedPreferences.Editor) {
            try {
                if (sApplyMethod != null) {
                    sApplyMethod.invoke(editor)
                    return
                }
            } catch (e: IllegalArgumentException) {
                e.printStackTrace()
            } catch (e: IllegalAccessException) {
                e.printStackTrace()
            } catch (e: InvocationTargetException) {
                e.printStackTrace()
            }
            editor.commit()
        }
    }
}
```

8.6 数据持久化处理——SQLite 数据库

对于一个实际应用而言，其注册账户数据实际上是可以很庞大的，同时对于一个在线新闻 app 而言，随着时间推移，其内部的新闻资讯更是庞大，其数据的存储不再适用于 SharedPreferences 存储处理，于是此时引入 SQLite 数据库进行数据管理。

相关技术代码及分析如下:

在 MainActivity 中定义 initDataUser 方法查询 user 数据库，显示账户相关信息:

```
    private fun initDataUser() {
    val userId = SPUtils[this, SPUtils.USER_ID, 0] as Int?
    val db = helper!!.writableDatabase
    val sql = "select * from user where id = ?"
    val cursor = db.rawQuery(sql, arrayOf(userId.toString()))
    if (cursor != null && cursor.columnCount > 0) {
        while (cursor.moveToNext()) {
```

```
                val dbId = cursor.getInt(0)
                val dbAccount = cursor.getString(1)
                val dbPassword = cursor.getString(2)
                val dbName = cursor.getString(3)
                val dbSex = cursor.getString(4)
                val dbPhone = cursor.getString(5)
                val dbAddress = cursor.getString(6)
                val dbPhoto = cursor.getString(7)
                mUser = User(dbId, dbAccount, dbPassword, dbName, dbSex, dbPhone, dbAddr
ess, dbPhoto)
            }
        }
        db.close()
        tvNickName!!.text = mUser!!.name
        Glide.with(this)
                .load(mUser!!.photo)
                .apply(headerRO.error(if ("男" == mUser!!.sex) R.drawable.ic_default_man else
R.drawable.ic_default_woman))
                .into(ivPhoto!!)
}
```

在 PersonActivity 中查询 user 数据库，更新账户相关信息:

```
    val sex = if (rgSex!!.checkedRadioButtonId == R.id.rb_man) "男" else "女" //性别
db.execSQL("update user set name = ?,phone=?,address=?,sex=? where id = ?", arrayOf<An
y>(nickName, phone, address, sex, mUser.id))
Toast.makeText(this@PersonActivity, "更新成功", Toast.LENGTH_SHORT).show()
db.close()
```

定义数据库的帮助类如下:

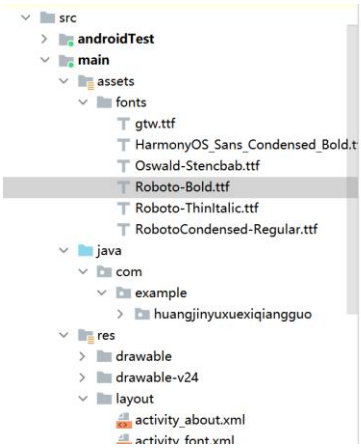
```
    class MySqliteOpenHelper(private val context: Context) : SQLiteOpenHelper(context, DB_
NAME, null, DB_VERSION) {
    // 在数据库首次创建的时候调用，创建表以及可以进行一些表数据的初始化
    override fun onCreate(db: SQLiteDatabase) {
        //创建表
        //_id 为主键并且自增长一般命名为_id
        val newsSql = "create table news(id integer primary key autoincrement,typeId,title,im
g,content,issuer,date)"
        val userSql = "create table user(id integer primary key autoincrement,account, passw
ord,name,sex, phone,address,photo)"
```

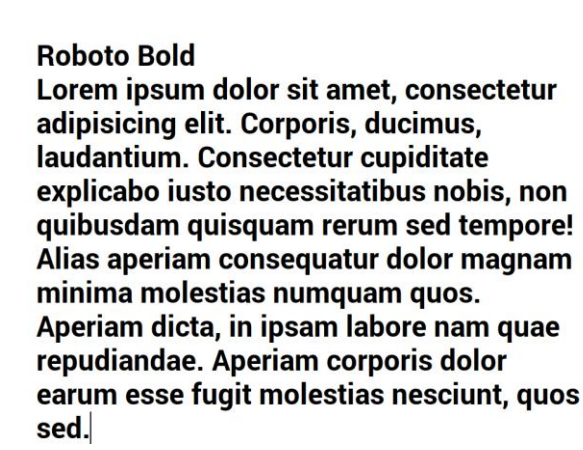
```
        db.execSQL(newsSql)
        db.execSQL(userSql)
    }
    //数据库升级的时候回调该方法，在数据库版本号 DB_VERSION 升级的时候才会调用
    override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int, newVersion: Int) {
    }
    //数据库打开的时候回调该方法
    override fun onOpen(db: SQLiteDatabase) {
        super.onOpen(db)
    }
    companion object {
        //数据库名字
        const val DB_NAME = "news.db"

        //数据库版本
        const val DB_VERSION = 1
    }
}
```

8.7 数据持久化处理——文件处理

1.字体文件

将字体文件放置在应用程序的 assets/fonts 目录下





初始化字体时读取资源文件，设置默认字体。

```
    private fun initFont() {
    ViewPump.init(ViewPump.builder()
            .addInterceptor(CalligraphyInterceptor(
                    CalligraphyConfig.Builder()
                            .setDefaultFontPath(SPUtils[this, "fonts", "fonts/Roboto-Bold.ttf
```

```
"] as String?)
                                .setFontAttrId(R.attr.fontPath)
                                .build()
                ))
                .build())
}
```

2.视频文件

将应用程序的运行视频放在了应用程序的 res/raw 目录下，然后利用 R.raw.video 方式读取资源文件:

```
var video = intArrayOf(R.raw.video)
```

再利用 Video.setVideoURI 对本地视频进行解析、播放:

```
lateinit var mVideo: VideoView
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_video)
    mVideo = findViewById<View>(R.id.video) as VideoView
    mVideo.setVideoURI(Uri.parse("android.resource://" + packageName + "/" + video[Rando
m().nextInt(video.size)]))
    val controller = MediaController(this)
    mVideo.setMediaController(controller)
    mVideo.start()
    initFont()
}
```

8.8 并发处理

并发处理技术在应用中主要体现在两个方面。

1.是使用 OKhttp3 工具时的多线程管理实现网络访问请求；2 是浏览新闻资讯时，UI 主线程用于展现网络新闻资讯，自定义线程利用 AsyncTask 异步执行，并结合 handler 机制用于记录当前页面的浏览时长。

相关技术代码及分析如下:

自定义 Handler 对象异步处理 message，在数据库中对用户的浏览积分进行更新，代码如下:

```
    private var mHandler: Handler = Handler {
    val db = helper!!.writableDatabase
    val userId = SPUtils[this@NewsDataDetailActivity, SPUtils.USER_ID, 0] as Int?
```

```
    val sql = "select * from user where id = ?"
    val cursor = db.rawQuery(sql, arrayOf(userId.toString()))
    var score = "0"
    if (cursor != null && cursor.columnCount > 0) {
        while (cursor.moveToNext()) {
            score = cursor.getString(8)
        }
    }
    try {
        val toInt = score.toInt()
        db.execSQL("update user set score = ? where id = ?", arrayOf<Any?>((toInt + 1).to
String(), userId))
    }catch (e:Exception){
    }
     db.close()
    false
}
```

在 NewsDataDetailActivity 中定义内部类 TimeTask 异步任务,重写 doInBackground()方法用于后台计时，重写 onProgressUpdate()用于 UI 线程更新时长。代码如下:

```
    inner class TimeTask : AsyncTask<Void?, Int?, Boolean>() {
    override fun doInBackground(vararg params: Void?): Boolean {
        while (isAlive) {
            Thread.sleep(1000)//当前线程休眠 1000ms
            publishProgress()
        }
        return true
    }
    override fun onProgressUpdate(vararg values: Int?) {
        mHandler.sendEmptyMessage(0);
        super.onProgressUpdate(*values)
    }
}
```

九、实验思考与总结

9.1 实验所遇问题及其解决方法

9.2 实验心得与体会

完成这个应用耗时较长，心中确实有较多体会。在实现这个应用的过程中，让我对 Android 开发的整体知识体系又有了一个更加深入的了解。之前都是每个章节的知识单独学习和实现，而这次综合实验却需要将所有的知识结合起来，当项目结构庞大起来，产生问题时更难找到问题所在，debug 的难度越大。

而且这次实验的要求较为宽泛，给了我们很大的发挥空间，如何将这些所学的知识变成实在的功能也需要我们查阅大量资料或者去感受相关的实际应用软件获得灵感。比如这次实验要求的并发处理技术，我开始并不知道怎么在新闻 app 中加入这个技术，后来加入的后台线程计时功能也是我在每天的“学习强国”中获得的启发，而且我的 UI 界面也在一定程度上参考了它的设计。因此移动应用开发这门课程也锻炼了我的想象力和创造力。

由于所学的技能浅薄，开发的应用还有很多技术不会运用，还有较多功能还待日后完善，但是完成应用的时刻，内心还是小有成就感，希望今后能在这条路上获得更多乐趣。

十、参考文献

[1] 郭霖. 第一行代码 Android(第 3 版)[M].人民邮电出版社，2019.

[2] 陈轶. Android 移动应用开发（微课版)[M].清华大学出版社,2022.

[3] 胡静. Android 平台的新闻客户端的设计与实现[J]. 信息与电脑(理论版),2021,33(24):121-123+149.

[4] 杨锰. 基于 Android 平台的新闻客户端的设计与实现[D].北京邮电大学,2021.

[5] 周明. 基于 Android 的手机网络新闻客户端设计与实现[D].东南大学,2018.

[6] Android RecyclerView 最全使用详解_Teacher.Hu 的博客-CSDN 博客_recyclerview

[7] Android 开发之设置 APP 全局字体_talentclass_ctt 的博客-CSDN 博客

[8] Recipes - OkHttp (square.github.io)

[9] okhttp3 (OkHttp 3.14.0 API) (square.github.io)

[10] Android OkHttp3 简介和使用详解_JackieZhengChina 的博客-CSDN 博客_android okh

ttp3

[11] Android：OkHttp 的理解和使用_JMW1407 的博客-CSDN 博客安卓 okhttpclient 使用

[12] syncTask 使用及解析_只管羊毛薅的博客-CSDN 博客_async task

[13] OkHttp3 简介和使用详解_胶泥座人的博客-CSDN 博客_okhttp3

[14] 好用的本地图片选择工具：PictureSelector_帅到不敢打代码的博客-CSDN 博客_pictureselector