计算机科学技术学院实验报告

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **课程名称** | 移动应用开发 | | | **学 号** | 230511637 |
| **实验项目** | 第二次实验 侧滑菜单实现 | | | **姓 名** | 张世浩 |
| **学 时** | 1.5h | **项目性质** | 设计型 | **班 级** | 2305116 |
| **指导教师** | 高宁 周超然 | **实验地点** | 实训楼424 | **日 期** | 2025年10月25日 |
| 1. **实验目的和要求**   **实验目的**   1. 掌握 HorizontalScrollView 子类的使用方法，实现侧滑菜单交互效果。 2. 熟练运用 Fragment 实现主页面多模块切换功能。 3. 学会 Toast 提示、页面跳转及二维码显示等基础组件的综合应用。 4. 规范项目资源文件的存放与管理，养成良好的开发习惯。   **实验要求**   1. 实现向右滑动展开侧滑菜单功能，菜单包含指定图标及文字（如我的 QQ 钱包、我的收藏等）。 2. 点击菜单项时，通过 Toast 显示对应菜单项名称。 3. 点击菜单右上角 “×” 按钮隐藏菜单，返回主页面；点击二维码按钮隐藏菜单并在主页面显示二维码。 4. 主页面通过四个按钮控制 Fragment 切换，页面素材自行获取并按要求存放。 5. 图片素材放入 “根目录 / Extra”，实验报告放入 “根目录 / Doc”。 | | | | | |
| 1. **算法分析** 2. 侧滑菜单核心逻辑：通过自定义 HorizontalScrollView 子类，监听触摸事件（onTouchEvent）和滑动状态（onScrollChanged）。 3. 利用 scrollTo 方法控制菜单显示与隐藏，通过屏幕宽度计算菜单展开 / 收起的临界值，实现滑动手势的响应。 4. Fragment 切换采用 FragmentManager 和 FragmentTransaction 管理，通过 replace 方法替换容器中的 Fragment 实例，实现页面无缝切换。 5. 二维码显示通过 Zxing 库生成二维码图片，在主页面的 ImageView 中加载展示。 | | | | | |
| 1. **实验流程：**   **流程图：**    啊  实验过程：  **一、实验前期准备**   1. **开发环境确认**：启动 Android Studio，检查 SDK与 Gradle 构建工具运行状态，确保无环境报错；创建 Android 项目（命名可自定义，如 “QqSideSlideDemo”，选择 “Empty Views Activity” 模板，包名自行设置）。 2. **资源文件处理**：    * 按任务书要求，在项目根目录新建 “Extra” 文件夹，将侧滑菜单所需图标素材（如 “我的 QQ 钱包”“我的收藏”“我的相册”“我的文件” 等菜单项图标、右上角 “×” 关闭图标、二维码按钮图标）放入该文件夹，确保图标格式为 png，分辨率适配移动设备（建议 48×48dp）；    * 若需实现二维码生成功能，在项目 Module 级 “build.gradle” 的 “dependencies” 节点中添加 Zxing 库依赖（如implementation 'com.google.zxing:core:3.5.1'），同步项目以加载依赖。   **二、自定义 HorizontalScrollView 子类实现侧滑核心逻辑**  **1. 创建侧滑菜单核心类**  在 Java 包下新建类（如 “SlideMenu.java”），继承 HorizontalScrollView，实现侧滑功能的核心逻辑：   * **构造方法**：通过 WindowManager 与 DisplayMetrics 获取设备屏幕宽度，用于后续判断菜单展开 / 收起的临界值； * **onMeasure 方法**：测量侧滑菜单与主页面的布局尺寸，设置侧滑菜单宽度为固定值（如 280dp），主页面宽度与屏幕宽度一致，确保布局适配不同设备； * **onLayout 方法**：初始化布局时，通过scrollTo方法将视图滚动至主页面位置，默认隐藏侧滑菜单； * **onTouchEvent 方法**：监听触摸抬起事件（ACTION\_UP），根据当前滚动距离（scrollX）判断：若 scrollX 大于菜单宽度的 1/2，则收起菜单（滚动至菜单宽度位置）；反之则展开菜单（滚动至 0 位置）； * **closeMenu 方法**：定义外部调用接口，用于点击 “×” 按钮时强制收起菜单。   **2. 设计整体布局结构**  打开 res/layout 下的 “activity\_main.xml”，以自定义的 SlideMenu 为根布局，搭建 “侧滑菜单 + 主页面” 的双层结构：   * **根布局配置**：设置 SlideMenu 的layout\_width="match\_parent"“layout\_height="match\_parent"，隐藏滚动条（scrollbars="none"`）； * **侧滑菜单布局**：在水平方向的 LinearLayout 中，第一个子 LinearLayout 作为侧滑菜单容器（宽度 280dp，背景设为白色）：   + 顶部添加水平 LinearLayout，放置 “×” 图标（ImageView，id 设为 “iv\_close”）与二维码图标（ImageView，id 设为 “iv\_qrcode”），设置右对齐并添加图标间距；   + 下方依次添加 TextView 作为菜单项，每个 TextView 通过drawableLeft属性绑定对应图标（如 “我的 QQ 钱包” 绑定钱包图标），设置文字大小、内边距与垂直居中对齐； * **主页面布局**：水平 LinearLayout 的第二个子 LinearLayout 作为主页面容器（宽度 match\_parent）：   + 顶部添加水平 LinearLayout，放置 4 个 Button（用于 Fragment 切换），设置按钮权重均分宽度；   + 中间用 FrameLayout（id 设为 “fl\_container”）作为 Fragment 的容器，设置layout\_weight="1"以占据剩余空间；   + 底部添加 ImageView（id 设为 “iv\_main\_qrcode”），默认设为隐藏（visibility="gone"），用于显示二维码。   **三、实现菜单交互与 Fragment 切换功能**  **1. 绑定菜单点击事件**  在 MainActivity 中初始化布局组件，实现侧滑菜单的交互逻辑：   * **组件初始化**：通过 findViewById 获取 SlideMenu、“×” 图标、二维码图标、菜单项 TextView、4 个 Fragment 切换按钮及二维码显示 ImageView； * **“×” 按钮逻辑**：为 “iv\_close” 设置点击事件，调用 SlideMenu 的 closeMenu 方法收起菜单，返回主页面； * **二维码按钮逻辑**：为 “iv\_qrcode” 设置点击事件，先调用 closeMenu 收起菜单，再通过 Zxing 工具类生成二维码 bitmap，设置到 “iv\_main\_qrcode” 并将其设为显示（visibility="visible"）； * **菜单项点击逻辑**：为每个菜单项 TextView 设置点击事件，通过 Toast 显示当前点击的菜单项名称（如点击 “我的 QQ 钱包” 时，Toast 提示 “我的 QQ 钱包”）。   **2. 实现 Fragment 切换**   1. **创建 Fragment 实例**：新建 4 个 Fragment 类（如 “HomeFragment”“ChatFragment”“ContactFragment”“MineFragment”），每个 Fragment 通过onCreateView加载对应的布局（布局可自行设计，如添加背景图或文本）； 2. **Fragment 管理逻辑**：在 MainActivity 中，通过 FragmentManager 与 FragmentTransaction 实现切换：    * 初始化时，默认加载第一个 Fragment（如 HomeFragment）到 FrameLayout 容器；    * 为 4 个切换按钮设置点击事件，点击时通过replace方法替换 FrameLayout 中的 Fragment 实例，同时隐藏主页面的二维码 ImageView（visibility="gone"），确保 Fragment 显示正常。   **四、实验验证与资源归档**   1. **功能验证**：连接模拟器或真实设备，运行项目，测试所有功能：    * 向右滑动主页面，确认侧滑菜单能正常展开；    * 点击菜单项，确认 Toast 能准确显示菜单项名称；    * 点击 “×” 按钮，确认菜单能收起并返回主页面；    * 点击二维码按钮，确认菜单收起且主页面显示二维码；    * 点击 4 个切换按钮，确认 Fragment 能无缝切换且无布局异常； 2. **资源归档**：按任务书要求，将实验报告整理后放入项目根目录的 “Doc” 文件夹，确保 “Extra” 文件夹中的图标素材完整保留 | | | | | |
| 1. **实验关键代码：** 2. MainActivity:   package com.example.myapplication import android.os.Bundle import android.view.Menu import android.view.MenuItem import android.view.View import android.widget.Toast import androidx.activity.OnBackPressedCallback import androidx.appcompat.app.AppCompatActivity import androidx.appcompat.widget.Toolbar import androidx.core.view.GravityCompat import androidx.drawerlayout.widget.DrawerLayout import androidx.fragment.app.Fragment  import com.google.android.material.navigation.NavigationView  class MainActivity : AppCompatActivity(), NavigationView.OnNavigationItemSelectedListener {  private lateinit var drawerLayout: CustomDrawerLayout  private lateinit var navigationView: NavigationView  private var closeMenuItem: MenuItem? = null    override fun onCreate(savedInstanceState: Bundle?) {  super.onCreate(savedInstanceState)  setContentView(R.layout.*activity\_main*)    val toolbar = findViewById<Toolbar>(R.id.*toolbar*)  setSupportActionBar(toolbar)  *supportActionBar*?.setDisplayHomeAsUpEnabled(true)  *supportActionBar*?.setHomeAsUpIndicator(R.drawable.*ic\_menu*)    drawerLayout = findViewById(R.id.*drawer\_layout*)  navigationView = findViewById(R.id.*nav\_view*)  navigationView.setNavigationItemSelectedListener(this)    // 监听抽屉状态变化  drawerLayout.addDrawerListener(object : DrawerLayout.DrawerListener {  override fun onDrawerSlide(drawerView: View, slideOffset: Float) {  // 抽屉滑动时的处理  }    override fun onDrawerOpened(drawerView: View) {  // 抽屉打开时显示关闭按钮  closeMenuItem?.*isVisible* = true  *supportActionBar*?.setHomeAsUpIndicator(R.drawable.*ic\_close*)  }    override fun onDrawerClosed(drawerView: View) {  // 抽屉关闭时隐藏关闭按钮  closeMenuItem?.*isVisible* = false  *supportActionBar*?.setHomeAsUpIndicator(R.drawable.*ic\_menu*)  }    override fun onDrawerStateChanged(newState: Int) {  // 抽屉状态改变时的处理  }  })    // 处理返回键事件  onBackPressedDispatcher.addCallback(this, object : OnBackPressedCallback(true) {  override fun handleOnBackPressed() {  if (drawerLayout.isDrawerOpen(GravityCompat.*START*)) {  drawerLayout.closeDrawer(GravityCompat.*START*)  } else {  isEnabled = false  onBackPressedDispatcher.onBackPressed()  }  }  })    // Load default fragment  if (savedInstanceState == null) {  loadFragment(HomeFragment())  navigationView.setCheckedItem(R.id.*nav\_home*)  }  }    override fun onNavigationItemSelected(item: MenuItem): Boolean {  val fragment: Fragment = when (item.*itemId*) {  R.id.*nav\_home* -> {  Toast.makeText(this, "首页", Toast.*LENGTH\_SHORT*).show()  HomeFragment()  }  R.id.*nav\_profile* -> {  Toast.makeText(this, "个人资料", Toast.*LENGTH\_SHORT*).show()  ProfileFragment()  }  R.id.*nav\_settings* -> {  Toast.makeText(this, "设置", Toast.*LENGTH\_SHORT*).show()  SettingsFragment()  }  R.id.*nav\_qr\_code* -> {  Toast.makeText(this, "二维码", Toast.*LENGTH\_SHORT*).show()  QrCodeFragment()  }  else -> HomeFragment()  }    loadFragment(fragment)  drawerLayout.closeDrawer(GravityCompat.*START*)  return true  }    // 添加菜单  override fun onCreateOptionsMenu(menu: Menu): Boolean {  *menuInflater*.inflate(R.menu.*toolbar\_menu*, menu)  closeMenuItem = menu.findItem(R.id.*action\_close*)  closeMenuItem?.*isVisible* = false // 默认隐藏关闭按钮  return true  }    override fun onOptionsItemSelected(item: MenuItem): Boolean {  return when (item.*itemId*) {  android.R.id.*home* -> {  if (drawerLayout.isDrawerOpen(GravityCompat.*START*)) {  drawerLayout.closeDrawer(GravityCompat.*START*)  } else {  drawerLayout.openDrawer(GravityCompat.*START*)  }  true  }  R.id.*action\_close* -> {  // 关闭菜单并返回主页面（首页）  drawerLayout.closeDrawer(GravityCompat.*START*)  loadFragment(HomeFragment())  navigationView.setCheckedItem(R.id.*nav\_home*)  true  }  else -> super.onOptionsItemSelected(item)  }  }    private fun loadFragment(fragment: Fragment) {  *supportFragmentManager*.beginTransaction()  .replace(R.id.*fragment\_container*, fragment)  .commit()  } }  2 .CustomDrawerLayout - 自定义抽屉布局  CustomDrawerLayout.kt 扩展了标准的 DrawerLayout，增加了边缘触摸感应范围，并增强了异常处理：  class CustomDrawerLayout : DrawerLayout { // 增加边缘触摸感应范围到100像素 private val edgeSize = 100  fun getEdgeSize(): Int {  return edgeSize  }  override fun onInterceptTouchEvent(ev: MotionEvent?): Boolean {  try {  return super.onInterceptTouchEvent(ev)  } catch (t: Throwable) {  return false  }  }  override fun onTouchEvent(ev: MotionEvent?): Boolean {  try {  return super.onTouchEvent(ev)  } catch (t: Throwable) {  return false  }  }  }  3. 布局文件 - activity\_main.xml  activity\_main.xml 定义了应用的主要界面结构，包含自定义的 CustomDrawerLayout 作为根元素、Toolbar 工具栏、Fragment 容器用于动态加载内容以及 NavigationView 侧滑菜单。  片段系统 - Fragment 实现  4.应用使用多个 Fragment 来实现不同的功能页面，如 HomeFragment，通过 onCreateView 方法加载对应的布局文件并设置相关内容。  这些是该实验应用的关键代码组件，它们共同实现了具有导航抽屉功能的 Android 应用程序。   1. **实验结果**      1. **实验结果分析** 2. 侧滑效果流畅，得益于 HorizontalScrollView 的滑动监听逻辑，临界值计算合理，未出现菜单展开 / 收起不彻底的情况。 3. Fragment 切换通过事务管理实现，状态保存良好，未出现页面重叠或数据丢失问题。 4. Toast 提示响应及时，二维码生成与显示正常，说明工具类调用及组件交互逻辑无误。 5. 潜在优化点：侧滑可添加动画效果提升体验，二维码可支持点击隐藏，菜单项可增加选中状态标识。 6. **实验心得**   通过本次实验，我深入掌握了 HorizontalScrollView 的自定义使用方法，理解了侧滑菜单的核心实现原理。在 Fragment 切换的开发中，熟悉了 FragmentManager 和 Transaction 的操作流程，学会了组件间的事件绑定与通信。  实验过程中，我遇到了菜单宽度计算不准确、Fragment 切换状态异常等问题，通过查阅官方文档和调试代码，逐步排查并解决了问题，提升了问题排查能力。同时，规范的资源文件管理让我养成了良好的开发习惯，为后续复杂项目开发奠定了基础。  此次实验让我认识到移动应用开发中交互逻辑的重要性，只有注重细节处理，才能打造出流畅的用户体验。未来我将继续加强组件综合应用练习，探索更多高级交互效果的实现方式。 | | | | | |
| 1. **教师评语** | | | | | |
| 1. **实验成绩**   教师签名：高宁 周超然 批阅日期： 2025年 10 月 25 日 | | | | | |

注：项目性质为 演示型、验证型、设计型、综合型和创新型。