

移动开发技术课程设计报告

题 目 手机文件管理系统

学生姓名 应涛 学号 190110910532

指导教师 王凌武

二级学院 信息管理与人工智能学院

专业名称 计算机专业

班 级 19计算机1班

2020年12月

**目录**

[1、设计任务书 1](#_Toc17570)

[2、本组课题及本人任务 1](#_Toc146)

[3、技术路线与实现方案 2](#_Toc615)

[4.、数据库设计 6](#_Toc26355)

[5、方法说明 8](#_Toc18202)

[6、模块介绍 …………………………………………………………………………………9](#_Toc18931)

[7、详细设计 10](#_Toc11952)

8、项目代码...........................................................................................................................16

## 设计任务书

设计一个安卓文件管理器，能够对文件或文件夹进行管理。包括复制、剪切、移动、删除、新建、搜索等操作； 能够支持对不同文件调用不同模块打开功能，比如：html文件调用系统浏览器打开，MP3文件能调用播放器打开等等，如找不到对应模块，显示“无法打开”即可； 支持后缀名显示和隐藏功能； 需以列表形式文件或目录，对文件和目录使用图片进行区分，可以采用类似windows电脑文件管理处理方式； 支持排序选择功能（比如：以时间排序、以大小排序）；能支持单个或多个文件或文件夹操作，比如：同时删除5个文件、同时删除一个目录下的文件和文件夹、同时复制粘贴五个文件等。

## 2、本组课题及本人任务

程序由本人独自设计完成。

## 3、技术路线与实现方案

#### 系统概况：

我完成的文件管理APP是基于Android平台和SQLite数据库。安卓手机只需装上APP， 并授予文件读写权限，就可以查看手机sd卡所存储的文件，并且可以新建文件或文件夹以及对已有文件进行一系列的操作。

#### ·文件管理系统所用技术介绍：

1）多种控件的使用

Toolbar：

替换了一直以来使用的ActionBar，使用更加方便，更容易来放置、实现动画以及控制。可以通过获取的toolbar的引用，调用相关API来实现更多操作，例如隐藏或显示操作栏。可以自定义很多功能，实用性很强。

FloatingActionMenu：

一个Floating Action Button（以下简称FAB）位于当前屏幕的最上层（覆盖于屏幕之上），用来执行最常见或者最重要的操作。它的UI表现形式为一个圆形容器和一个位于圆圈中心的icon。



图表 1 FAB示意图

他的使用非常简单只需要在布局文件中引入控件即可，可以通过定义属性来方便的进行操作，常用的属性有位置，阴影，点击事件，展开按钮个数，大小等。不会和原有的布局产生冲突，兼容性很好。

XCRoundImageView自定义控件：

自定义的圆形ImageView，可以直接当组件在布局中使用。通过该组件实现一个圆形的头像框用于个人介绍界面。



图表 2 自定义组件展示

LinearLayout灰色蒙层：

在主界面的布局中添加一个不可见的全局LinearLayout用作灰色蒙层，设置其背景色为略微透明的浅灰色，当使用悬浮按钮时显示灰色蒙层可以将焦点都聚集在悬浮按钮处，在点击别处时让灰色蒙层重新不可见并收起悬浮按钮，优化了用户的交互体验。

2）RecyclerView+Adapter+ViewHolder

RecyclerView是Android一个更强大的控件,其不仅可以实现和ListView同样的效果,还有优化了ListView中的各种不足。其可以实现数据纵向滚动,也可以实现横向滚动(ListView做不到横向滚动)。

用户滑动屏幕切换视图时，上一个视图会回收利用，RecyclerView所做的就是回收再利用，循环往复。ViewHolder的主要任务时容纳View视图。Adapter主要的任务是创建ViewHolder和将数据绑定到ViewHolder上。在ViewHolder设置各种状态的操作，如多选模式，全选模式，单击或长按文件，文件夹等操作。

3）异步操作

自定义异步任务，在进行搜索时开启异步任务，后台静默搜索，搜索到的结果通过 UI 线程添加到视图上。

4）SQL数据库+GreenDao框架

SQLite是一款轻量级的开源的嵌入式数据库，由D.Richard Hipp在2000年发布。SQLite使用方便，性能出众，广泛应用于消费电子、医疗、工业控制、军事等各种领域。

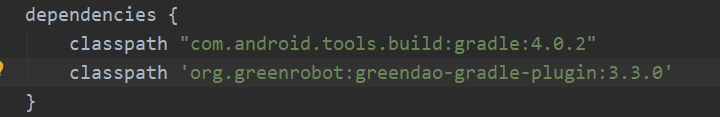
它具有以下特点：

* 体积小：最低只需要几百K的内存就可以运行。
* 性能高：对数据库的访问性能很高，其运行速度比Mysql等开源数据库要快很多。
* 可移植性强：能支持各种32位和64位体系的硬件平台，也能在Windows、Linux、BSD、Mac OS、Solaries等软件平台中运行。
* SQL支持：SQLite支持ANSI SQL92中的大多数标准，提供了对子查询、视图、触发器等机制的支持。

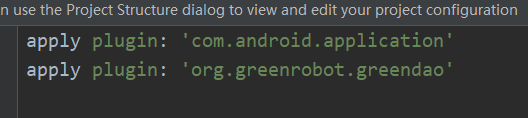
GreenDao框架：

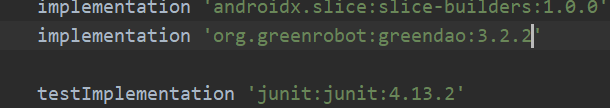
greenDAO 是适用于 Android 的轻量级快速 ORM 框架，可将对象映射到 SQLite 数据库中。 并且针对 Android 进行了高度的优化，greenDAO 提供了出色的性能，并占用了最少的内存，优点在性能上（可能是 Android 上最快的 ORM 框架）;易用性上（提供强大并且简洁明了的 API）；轻量（最小的内存消耗与小于 150KB 的库大小）。

Greendao的使用操作十分方便，使用前先引入对应的包，需要注意安卓版本问题。



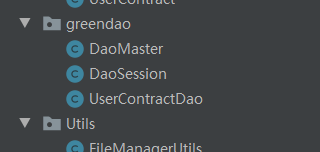
图表 3 外层build.gradle





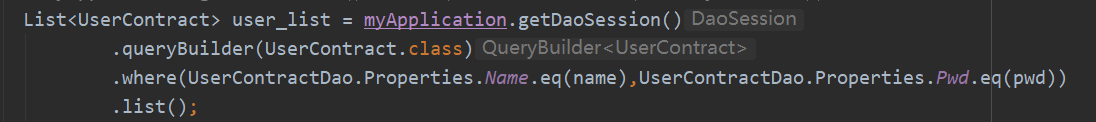
图表 4 app：build.gradle

引入完成后创建实体类，点击Build → Make Project会自动生成代码。生成结果如下：



图表 5 自动生成的greendao包

生成完毕后就可以通过该框架方便的操作SQlite数据库。



图表 6 使用框架进行查询

5) GestureDetector 手势识别器:

通过手势识别实现左滑或右滑出现个人介绍界面，通过OnGestureListener监听页面的手势变化，通过onFling()的四个参数来实现滑动效果，通过设置手势滑动最小长度和手势滑动最小速度实现高精度的识别手势。OnFling的四个参数意思分别为手势起点的移动事件，当前手势点的移动事件，每秒x轴方向移动的像素每秒y轴方向移动的像素。



通过计算可以准确的识别出手势是左滑还是右滑，当然也可以识别上滑和下滑，在对应的条件里写入事件即可。我在左滑的条件里写入了一个显示dialog的事件，在dialog中展示自己的个人信息。

#### 技术实现

#### ·开发环境

（1）开发工具：Android studio

1）集成开发环境（IDE）:

android +JDK8+Android SDK

2）采用JAVA开发。

（2）操作系统 ：Windows 11

（3）数据库：SQLite+greendao

## 4数据库设计

表4.1ser表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| User | Userid | Long | 64 | 非空 |
|  | username | varchar | 20 | 非空 |
|  | password | varchar | 20 | 非空 |

## 5.方法说明

表5. 方法表

|  |  |  |
| --- | --- | --- |
| 包 com.example.filemanager.Activity.BaseActivity | | |
| 方法名 | 功能 | 备注（须输入的参数） |
| onCreate | BaseActivity初始化 | 无 |
| setBottomSheet | 设置底部试图 | 无 |
| setFloatingMenu | 设置悬浮按钮 |  |
| onCreateOptionsMenu | 填充菜单；将项目添加到操作栏 | Menu menu |
| onOptionsItemSelected | 选择操作 | MenuItem item |
| onBackPressed | 根据当前状态确定 back 键的功能 |  |
| onRequestPermissionsResult | 确认权限 | requestCode，permissions |
| init | 初始化页面 |  |
| checkPermission | 判断权限 |  |
| setSelectModeShow | 设置多选模式下的界面 | isSelectMode |
| setFileViewSort | 设置文件排序 | sort |
| closeFloatingMenu | 关闭悬浮按钮 |  |
| pasteFile | 粘贴文件 |  |
| cutOrCopyFile | 剪切或复制文件，剪切文件则将文件从 UI 中删除 | isCut |
| deleteFile | 删除文件 |  |
| createFileOrDir | 创建文件或文件夹 | Path，isDirectory |
|  |  |  |
| 包 com.example.filemanager.Activity.Mainactivity | | |
| 方法名 | 功能 | 备注（须输入的参数） |
| onOptionsItemSelected | 选择页面 | MenuItem item |
| onFling | 手势识别函数 |  |
| 包 com.example.filemanager.Activity.Searchactivity | | |
| onOptionsItemSelected | 模式选择 | MenuItem item |
| onBackPressed | 返回 |  |
| onDestroy | 返回上一页前结束搜索任务 |  |
| search | 搜索 |  |
| SearchTask | 自定义异步任务 |  |

## 6．模块介绍

**1) 用户管理模块**

用户只有在登陆状态才有权限对文件进行读写操作，若不在登录状态则无法进行写文件操作，如新建文件夹或新建文件。

1. **文件新建模块**

·新建文件:通过新建文件写上对应后缀名将自动转换为对应文件，若后缀名错误则会显示位置文件。

·新建文件夹:新建一个文件夹，位置根据自己选定的排序规则决定

1. **顶部栏模块**

·用户按钮:点击用户按钮进行登录获得写文件权限

·搜索按钮:点击搜索按钮跳转搜索界面，根据输入的文字进行模糊匹配，在后台异步进行实时将文件在UI界面展示

·扩展按钮：可选择全选，显示后缀名 ，根据名称，大小，时间将文件进行排序。

1. **底部模块**

长按文件将弹出底部框，文件左侧将出现勾选框，支持多选，底部由复制，粘贴，剪切，删除按钮，对选中的全部文件生效。

## 7、详细设计

**1) 登录**

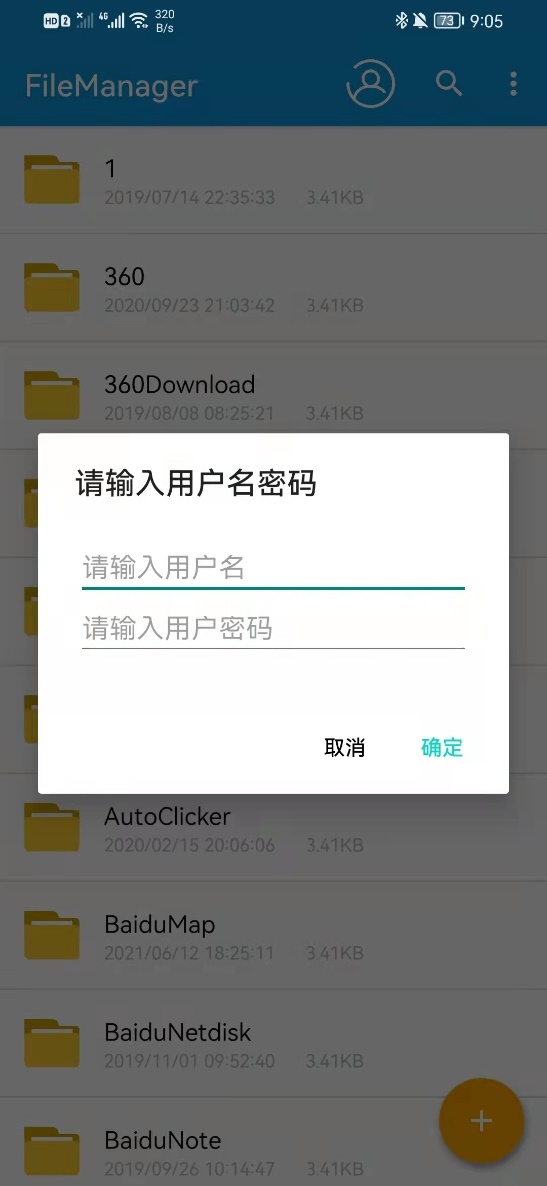


图7.1 用户登录

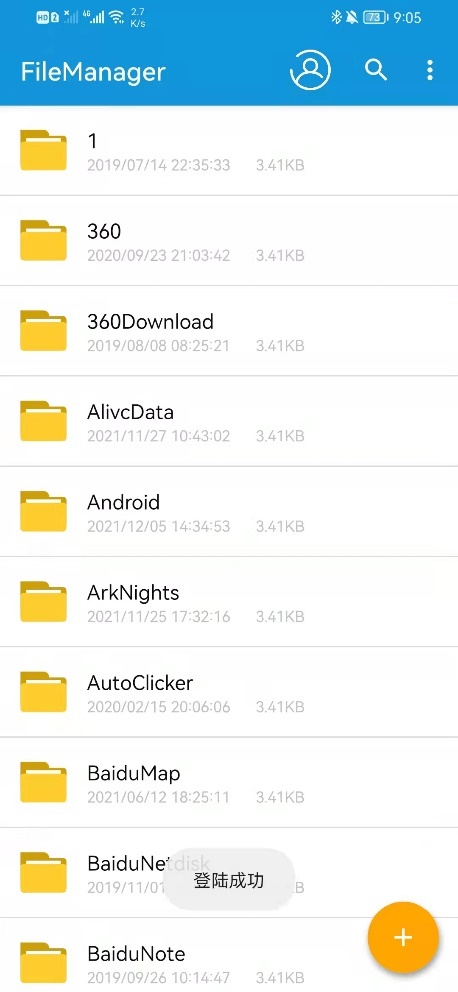


图7.2 登录成功

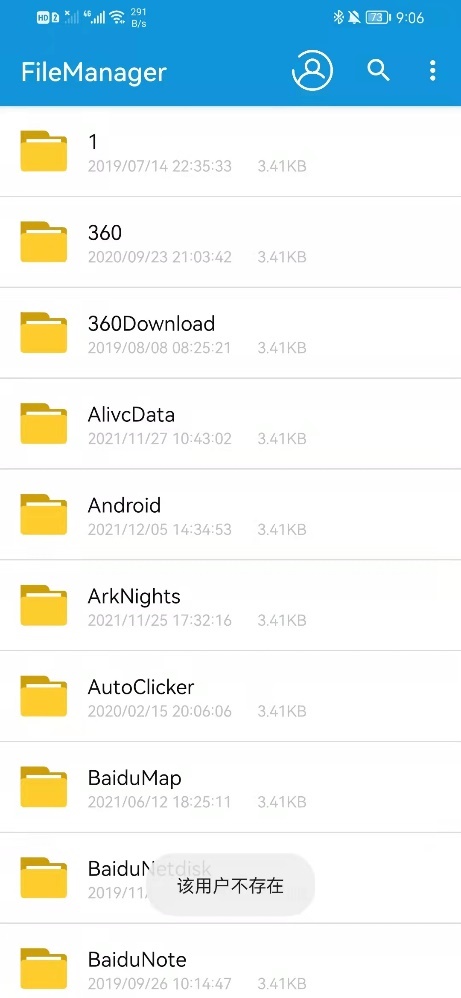


图7.3 登录失败

1. **新建**

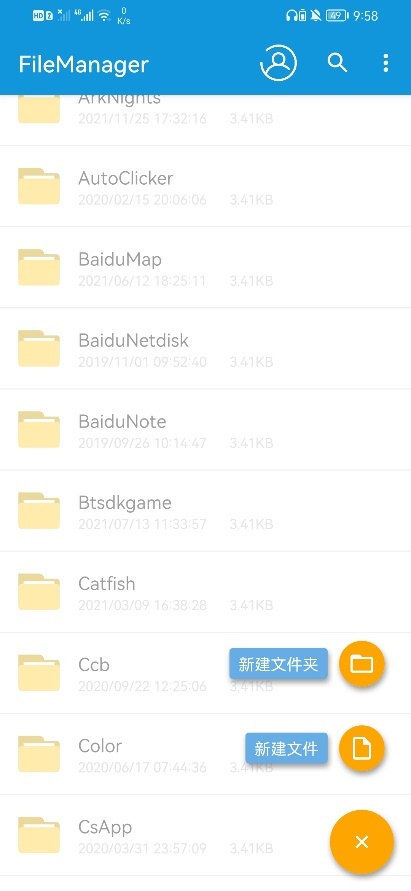


图7.4 新建按钮

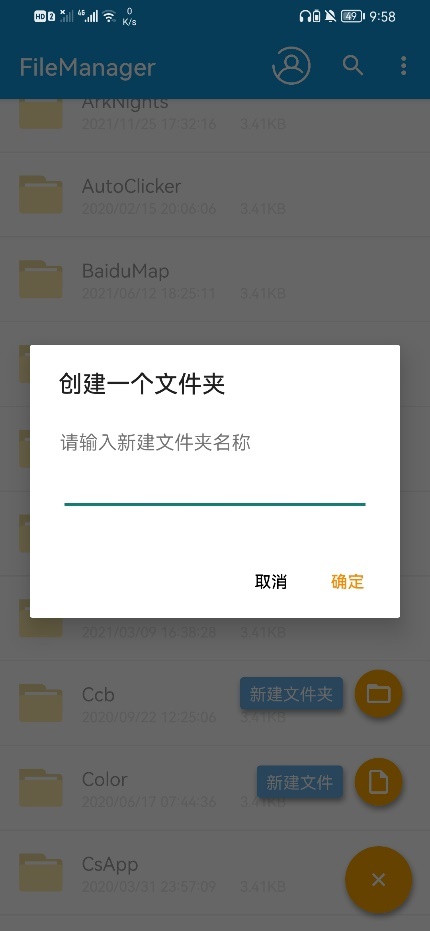


图7.5 新建文件夹

1. **顶部栏**

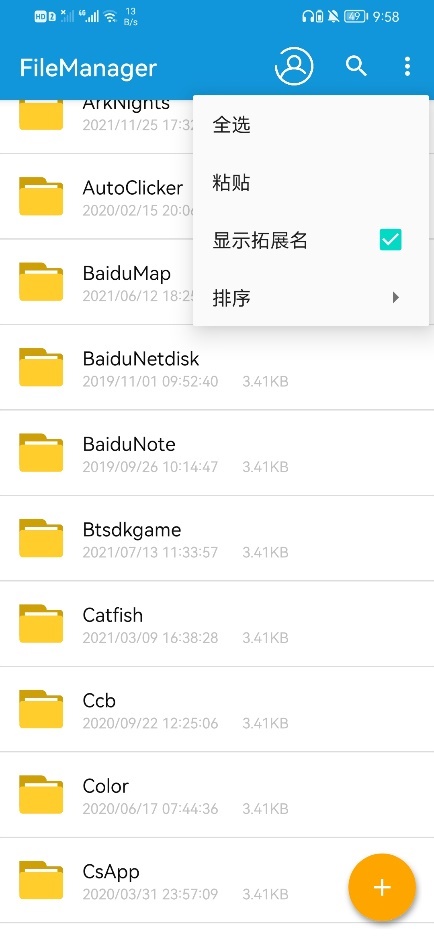


图7.6 顶部栏

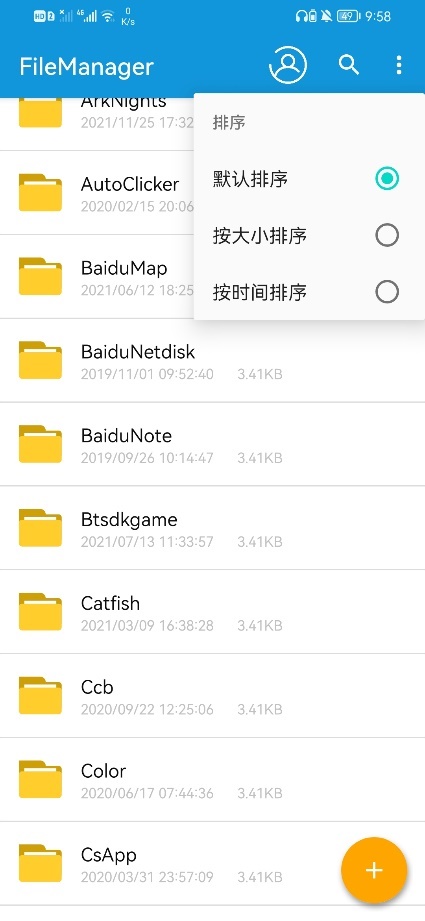


图7.7 排序规则

1. **主界面**

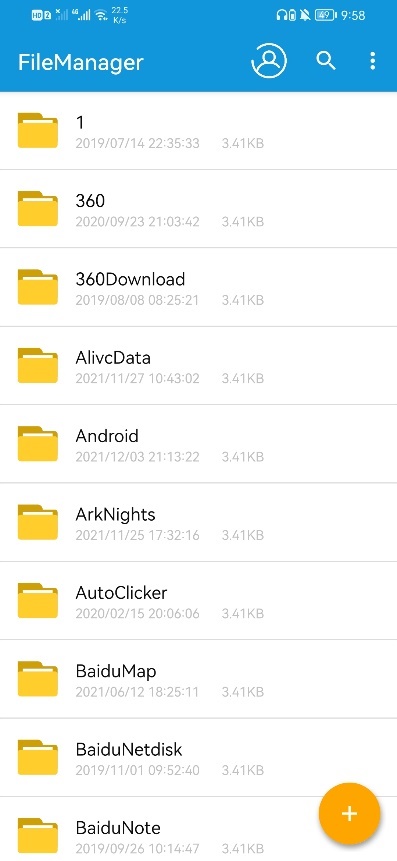


图7.8 主界面

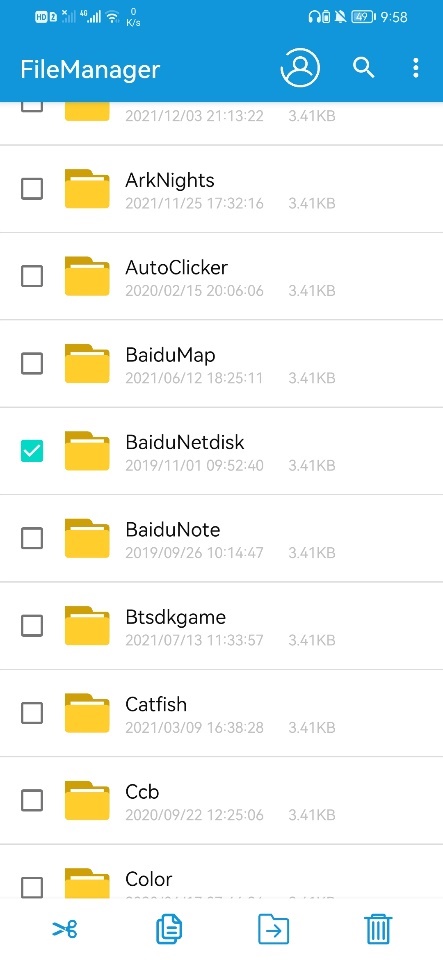


图7.9 长按模式

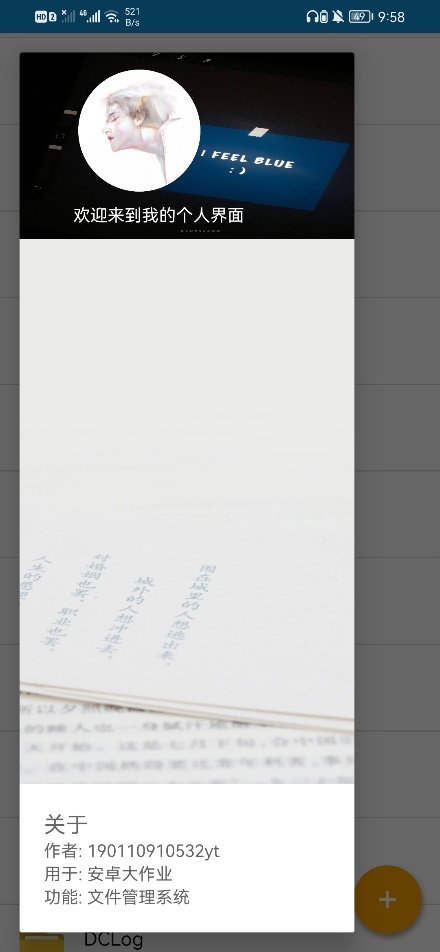


图7.10 左滑个人界面

1. **搜索界面**

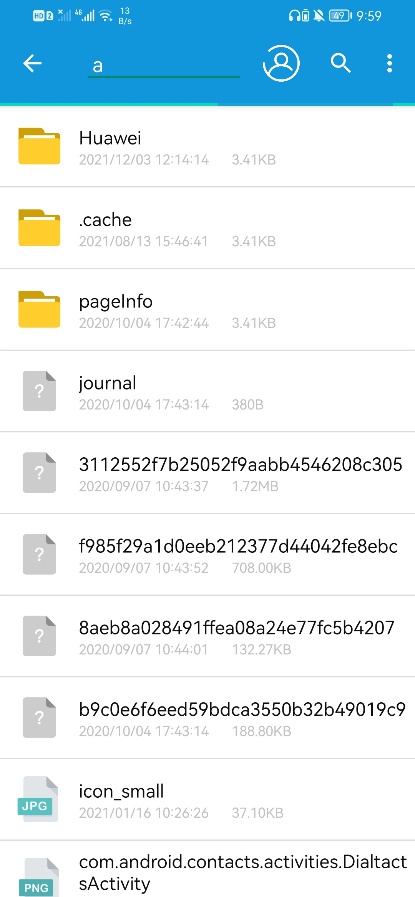


图7.11 搜索界面

**8.项目代码**

**baseActivity**

1. **public** **abstract** **class** BaseActivity **extends** AppCompatActivity {
2. **public** FloatingActionMenu fab;                     //  右下悬浮按钮
3. **private** LinearLayout greyCover;                     //  灰色蒙层
4. **public** RecyclerView recyclerView;
5. /\*\* RecyclerView
6. \* 可以通过设置LayoutManager来快速实现listview、gridview、瀑布流的效果，而且还可以设置横向和纵向显示，
7. \* 添加动画效果也非常简单(自带了ItemAnimation，可以设置加载和移除时的动画，方便做出各种动态浏览的效果)
8. \*/
9. **public** LinearLayout bottomSheetLayout;
10. **public** BottomSheetBehavior bottomSheetBehavior;     //  底部操作栏
12. **private** **final** String[] permissions = **new** String[]{  //  应用所需权限
13. Manifest.permission.WRITE\_EXTERNAL\_STORAGE, //写
14. Manifest.permission.READ\_EXTERNAL\_STORAGE   //读
15. };
16. **private** **final** List<String> mPermissionList = **new** ArrayList<>(); //  已同意的权限
17. **protected** String path;            //  当前文件树结点的路径
18. **public** FileViewAdapter adapter;
19. **public** List<FileView> fileList;   //  文件列表
20. **public** MyApplication myDaoApplication; //获取daoSession
22. @Override
23. **protected** **void** onCreate(Bundle savedInstanceState) {
24. **super**.onCreate(savedInstanceState);
25. //  设置布局
26. setContentView(R.layout.activity\_main);
27. //  开启Toolbar功能
28. Toolbar toolbar = findViewById(R.id.toolbar);
29. setSupportActionBar(toolbar);
30. //  文件列表视图初始化
31. recyclerView = findViewById(R.id.file\_container);
32. LinearLayoutManager layoutManager = **new** LinearLayoutManager(**this**);
33. recyclerView.setLayoutManager(layoutManager);
34. recyclerView.addItemDecoration(**new** DividerItemDecoration(**this**, DividerItemDecoration.VERTICAL));//添加item之间的分割线
36. //  底部操作栏初始化
37. bottomSheetLayout = findViewById(R.id.bottomSheetLayout);
38. bottomSheetBehavior = BottomSheetBehavior.from(bottomSheetLayout);
40. //  检查权限，若权限全同意则执行 init 方法
41. checkPermission();
42. //  设置底部操作栏各个按钮的点击回调
43. setBottomSheet();
44. //  设置右下悬浮按钮的点击回调
45. setFloatingMenu();
46. }
48. **private** **void** setBottomSheet() {
49. findViewById(R.id.bottom\_delete).setOnClickListener(view -> deleteFile());
50. findViewById(R.id.bottom\_cut).setOnClickListener(view -> cutOrCopyFile(**true**));
51. findViewById(R.id.bottom\_copy).setOnClickListener(view -> cutOrCopyFile(**false**));
52. findViewById(R.id.bottom\_paste).setOnClickListener(view -> pasteFile());
53. }
55. **private** **void** setFloatingMenu() {
56. fab = findViewById(R.id.fab);
57. greyCover = findViewById(R.id.grey\_cover);
59. fab.setOnMenuButtonClickListener(view -> {
60. setSelectModeShow(**false**);
61. **if** (fab.isOpened()) {
62. greyCover.setVisibility(View.GONE);  //设置灰色蒙层不可见
63. } **else** {
64. greyCover.setVisibility(View.VISIBLE);  //设置灰色蒙层可见
65. }
66. fab.toggle(**true**);
67. });
69. FloatingActionButton createDirBtn = findViewById(R.id.create\_dir);
70. FloatingActionButton createFileBtn = findViewById(R.id.create\_file);
71. createDirBtn.setOnClickListener(view -> createFileOrDirBtnCallback(**true**));
72. createFileBtn.setOnClickListener(view -> createFileOrDirBtnCallback(**false**));
74. //  点击灰色蒙层时关闭 FloatingActionMenu
75. greyCover.setOnClickListener(view -> closeFloatingMenu());
76. }
78. @Override
79. **public** **boolean** onCreateOptionsMenu(Menu menu) {
80. // 填充菜单；将项目添加到操作栏（如果存在）。
81. getMenuInflater().inflate(R.menu.menu\_main, menu);
82. **return** **true**;
83. }
85. @Override
86. **public** **boolean** onOptionsItemSelected(MenuItem item) {
87. **int** id = item.getItemId();
89. **switch** (id) {
90. **case** R.id.show\_ext:
91. item.setChecked(!item.isChecked());
92. adapter.setShowExt(item.isChecked());
93. **return** **true**;
94. **case** android.R.id.home:
95. **if** (!adapter.leaveSelectMode()) {
96. **if** (fab.isOpened()) {
97. closeFloatingMenu();
98. } **else** {
99. finishAfterTransition();
100. }
101. }
102. **return** **true**;
103. **case** R.id.sort\_by\_default:
104. **if** (!item.isChecked()) {
105. setFileViewSort(GetFilesUtils.SORT\_BY\_DEFAULT);
106. item.setChecked(**true**);
107. }
108. **return** **true**;
109. **case** R.id.sort\_by\_size:
110. **if** (!item.isChecked()) {
111. setFileViewSort(GetFilesUtils.SORT\_BY\_SIZE);
112. item.setChecked(**true**);
113. }
114. **return** **true**;
115. **case** R.id.sort\_by\_time:
116. **if** (!item.isChecked()) {
117. setFileViewSort(GetFilesUtils.SORT\_BY\_TIME);
118. item.setChecked(**true**);
119. }
120. **return** **true**;
121. **case** R.id.paste:
122. pasteFile();
123. **return** **true**;
124. **case** R.id.select\_all:
125. adapter.selectAll();
126. **return** **true**;
127. }
128. **return** **super**.onOptionsItemSelected(item);
129. }
131. @Override
132. **public** **void** onBackPressed() {
133. //  根据当前状态确定 back 键的功能
134. **if** (!adapter.leaveSelectMode()) {
135. **if** (fab.isOpened()) {
136. closeFloatingMenu();
137. } **else** {
138. **super**.onBackPressed();
139. }
140. }
141. }
143. @Override
144. **public** **void** onRequestPermissionsResult(**int** requestCode, @NonNull String[] permissions, @NonNull **int**[] grantResults) {
145. **if** (requestCode == 1) {
146. **if** (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION\_GRANTED && grantResults[1] == PackageManager.PERMISSION\_GRANTED) {
147. init();
148. } **else** {
149. Toast.makeText(**this**, "Permission Denied", Toast.LENGTH\_LONG).show();
150. finishAfterTransition();
151. }
152. }
153. }
155. **protected** **void** init() {
156. Intent intent = getIntent();
157. path = intent.getStringExtra("path");
158. //  path 为 null，代表当前节点为根节点，path 更改为根路径
159. **if** (path == **null**) {
160. path = GetFilesUtils.getInstance().getBasePath();
161. }
163. //  BFS遍历一层节点并渲染
164. **this**.fileList = GetFilesUtils.getInstance().getChildNode(path);
165. adapter = **new** FileViewAdapter(**this**.fileList);
166. recyclerView.setAdapter(adapter);
167. }

170. **private** **void** checkPermission() {
171. **for** (String permission : permissions) {
172. **if** (ContextCompat.checkSelfPermission(**this**, permission) !=
173. PackageManager.PERMISSION\_GRANTED) {
174. mPermissionList.add(permission);
175. }
176. }
178. //  未同意权限则请求权限，否则直接执行 init 方法
179. **if** (!mPermissionList.isEmpty()) {
180. String[] permissions1 = mPermissionList.toArray(**new** String[0]);
181. ActivityCompat.requestPermissions(**this**, permissions1, 1);
182. } **else** {
183. init();
184. }
185. }
187. /\*\*
188. \* 设置多选模式下的界面
189. \* @param isSelectMode
190. \*/
191. **public** **void** setSelectModeShow(**boolean** isSelectMode) {
192. **if** (isSelectMode) { //设置底部栏可见，隐藏悬浮按钮
193. fab.setVisibility(View.GONE);
194. bottomSheetLayout.setVisibility(View.VISIBLE);
195. bottomSheetBehavior.setState(BottomSheetBehavior.STATE\_EXPANDED);
196. } **else** **if**(!isSelectMode){//设置底部栏不可见，显示悬浮按钮
197. fab.setVisibility(View.VISIBLE);
198. bottomSheetLayout.setVisibility(View.GONE);
199. bottomSheetBehavior.setState(BottomSheetBehavior.STATE\_COLLAPSED);
200. }
201. }
203. /\*\*
204. \* 设置文件排序
205. \* @param sort
206. \*/
207. **private** **void** setFileViewSort(String sort) {
208. Collections.sort(**this**.fileList, GetFilesUtils.getInstance().fileOrder(sort));
209. adapter.notifyDataSetChanged();
210. }
212. **private** **void** closeFloatingMenu() {
213. fab.close(**true**);
214. greyCover.setVisibility(View.GONE);
215. }
217. /\*\*
218. \* 剪切或复制文件，剪切文件则将文件从 UI 中删除
219. \*
220. \* @param isCut 是否为剪切操作
221. \*/
222. **private** **void** cutOrCopyFile(**boolean** isCut) {
223. Set<FileView> selected = adapter.getSelectSet();
224. List<FileView> fileList = **new** ArrayList<>(selected);
225. **if** (isCut) {
226. FileManagerUtils.Instance.cut(fileList);
227. Toast.makeText(**this**, "已剪切~请移动到目标目录下粘贴", Toast.LENGTH\_SHORT).show();
228. **this**.fileList.removeAll(fileList);
229. } **else** {
230. FileManagerUtils.Instance.copy(fileList);
231. Toast.makeText(**this**, "已复制~请移动到目标目录下粘贴", Toast.LENGTH\_SHORT).show();
232. }
233. adapter.notifyOperationFinish();
234. }
236. **private** **void** pasteFile() {
237. **if** (FileManagerUtils.Instance.isClipBoardEmpty()) {
238. Toast.makeText(**this**, "剪切板为空~", Toast.LENGTH\_SHORT).show();
239. **return**;
240. }
241. **try** {
242. List<FileView> result = FileManagerUtils.Instance.paste(**new** File(path));
243. fileList.addAll(result);
244. Toast.makeText(**this**, "粘贴成功~", Toast.LENGTH\_SHORT).show();
245. adapter.notifyOperationFinish();
246. } **catch** (IOException e) {
247. e.printStackTrace();
248. }
249. }
251. /\*\*
252. \* 删除文件前弹窗确认操作，确认后执行删除操作
253. \*/
254. **private** **void** deleteFile() {
255. **final** Set<FileView> selected = adapter.getSelectSet();
256. **final** Context context = **this**;
257. **new** AlertDialog.Builder(**this**)
258. .setTitle("删除文件")
259. .setMessage("你确定要删除" + selected.size() + "个文件/文件夹？此操作无法撤销。")
260. .setNegativeButton("确定", (dialogInterface, i) -> {
261. //  遍历勾选文件列表，逐个删除文件
262. **for** (FileView fileView : selected) {
263. **if** (!FileManagerUtils.Instance.delete(fileView.getFile())) {
264. Toast.makeText(context, "删除文件失败", Toast.LENGTH\_SHORT).show();
265. **return**;
266. }
267. fileList.remove(fileView);
268. }
269. adapter.notifyOperationFinish();
270. Toast.makeText(context, "删除文件成功", Toast.LENGTH\_SHORT).show();
271. })
272. .setPositiveButton("取消", (dialogInterface, i) -> {
274. }).show();
275. }

278. /\*\*
279. \* 创建文件或文件夹
280. \* @param path
281. \* @param isDirectory
282. \*/
283. **private** **void** createFileOrDir(String path, **boolean** isDirectory) {
284. closeFloatingMenu();
285. **if** (isDirectory) {
286. **boolean** result = FileManagerUtils.Instance.createDirectory(path);
287. **if** (result) {
288. Toast.makeText(**this**, "创建成功", Toast.LENGTH\_SHORT).show();
289. } **else** {
290. Toast.makeText(**this**, "创建失败，可能存在同名文件夹", Toast.LENGTH\_SHORT).show();
291. **return**;
292. }
293. } **else** {
294. **try** {
295. **boolean** result = FileManagerUtils.Instance.createFile(path);
296. **if** (result) {
297. Toast.makeText(**this**, "创建成功", Toast.LENGTH\_SHORT).show();
298. } **else** {
299. Toast.makeText(**this**, "创建失败可能存在同名文件", Toast.LENGTH\_SHORT).show();
300. **return**;
301. }
302. } **catch** (IOException e) {
303. Toast.makeText(BaseActivity.**this**, "创建失败, 原因是：" + e.getMessage(), Toast.LENGTH\_SHORT).show();
304. **return**;
305. }
306. }
307. //  创建成功，更新 UI，并滑动到新建的文件/文件夹位置
308. FileView fileView = **new** FileView(**new** File(path));
309. fileList.add(fileView);
310. adapter.notifyOperationFinish();
311. **int** position = fileList.indexOf(fileView);
312. **if** (position != -1) {
313. recyclerView.scrollToPosition(position);
314. LinearLayoutManager mLayoutManager =
315. (LinearLayoutManager) recyclerView.getLayoutManager();
316. mLayoutManager.scrollToPositionWithOffset(position, 0);
317. }
318. }
320. //  创建文件/文件夹按钮的回调，创建一个弹窗
321. **private** **void** createFileOrDirBtnCallback(**boolean** isDirectory) {
322. **final** View dialogView = LayoutInflater.from(BaseActivity.**this**)
323. .inflate(R.layout.input\_dialog, **null**);
324. ((TextView) dialogView.findViewById(R.id.dialog\_tip)).setText(isDirectory ? "请输入新建文件夹名称" : "请输入新的文件名");
326. **final** EditText editText = dialogView.findViewById(R.id.dialog\_input);
327. editText.requestFocus();
329. **final** AlertDialog dialog =
330. **new** AlertDialog.Builder(BaseActivity.**this**).setTitle(isDirectory ? "创建一个文件夹" : "创建一个空白文件").setView(dialogView).setPositiveButton(
331. "确定", **null**).setNegativeButton("取消", (dialog1, which) -> {
332. }).create();
333. dialog.setOnShowListener(dialogInterface -> {
334. Button positiveBtn = dialog.getButton(DialogInterface.BUTTON\_POSITIVE);
335. positiveBtn.setOnClickListener(v -> {
336. String name = editText.getText().toString().trim();
337. **if** (TextUtils.isEmpty(name)) {
338. Toast.makeText(BaseActivity.**this**, "名称不能为空", Toast.LENGTH\_SHORT).show();
339. } **else** {
340. String newPath = path + "/" + name;
341. createFileOrDir(newPath, isDirectory);
342. dialog.dismiss();
343. }
344. });
345. });
346. dialog.show();
347. dialog.getButton(AlertDialog.BUTTON\_POSITIVE).setTextColor(getResources().getColor(R.color.orangeDark));
348. dialog.getButton(DialogInterface.BUTTON\_NEGATIVE).setTextColor(Color.BLACK);
349. }
350. }

**mainActivity**

1. **public** **class** MainActivity **extends** BaseActivity {
3. **private** **static** **final** String TAG = "MainActivity";
4. **private** **final** GestureDetector gdt = **new** GestureDetector((GestureDetector.OnGestureListener) **new**  GestureListener());
6. @Override
7. **protected** **void** onCreate(Bundle savedInstanceState) {
8. **super**.onCreate(savedInstanceState);
9. **final** RecyclerView recyclerView = (RecyclerView) findViewById(R.id.file\_container);
10. recyclerView.setOnTouchListener(**new** View.OnTouchListener() {
11. @Override
12. **public** **boolean** onTouch(**final** View view, **final** MotionEvent event) {
13. **return** gdt.onTouchEvent(event);
14. }
15. });
16. }
18. @Override
19. **public** **boolean** onOptionsItemSelected(MenuItem item) {
20. //  进入搜索页面
21. **if** (item.getItemId() == R.id.search) {
22. Intent intent = **new** Intent(**this**, SearchActivity.**class**);
23. intent.putExtra("path", path);
24. startActivity(intent);
25. **return** **true**;
26. }
27. //进入用户验证界面
28. **if** (item.getItemId() == R.id.changeUser) {
29. MyApplication myApplication = (MyApplication) getApplication(); //获取MyApplication得到session
30. **final** View dialogView = LayoutInflater.from(MainActivity.**this**)
31. .inflate(R.layout.cgperson\_dialog, **null**);
32. **final** EditText editText = dialogView.findViewById(R.id.dialog\_input1);
33. **final** EditText editText2 = dialogView.findViewById(R.id.dialog\_input2);
35. editText.requestFocus();
37. **final** AlertDialog dialog =
38. **new** AlertDialog.Builder(MainActivity.**this**).setTitle("请输入用户名密码").setView(dialogView).setPositiveButton(
39. "确定", **null**).setNegativeButton("取消", (dialog1, which) -> {
40. }).create();
41. dialog.setOnShowListener(dialogInterface -> {
42. Button positiveBtn = dialog.getButton(DialogInterface.BUTTON\_POSITIVE);
43. positiveBtn.setOnClickListener(v -> {
44. String name = editText.getText().toString().trim();
45. String pwd = editText2.getText().toString().trim();
46. **if** (TextUtils.isEmpty(name)||TextUtils.isEmpty(pwd)) {
47. Toast.makeText(MainActivity.**this**, "用户名和密码不能为空", Toast.LENGTH\_SHORT).show();
48. } **else** {
49. //            myApplication.getDaoSession().insert(new UserContract(null,"yt","123"));
50. List<UserContract> user\_list = myApplication.getDaoSession()
51. .queryBuilder(UserContract.**class**)
52. .where(UserContractDao.Properties.Name.eq(name),UserContractDao.Properties.Pwd.eq(pwd))
53. .list();
54. **if**(user\_list.size()<=0){
55. fab.setVisibility(View.GONE);
56. bottomSheetLayout.setVisibility(View.GONE);
57. bottomSheetBehavior.setState(BottomSheetBehavior.STATE\_COLLAPSED);
58. Toast.makeText(**this**, "该用户不存在", Toast.LENGTH\_SHORT).show();
59. }**else** {
60. fab.setVisibility(View.VISIBLE);
61. bottomSheetLayout.setVisibility(View.GONE);
62. bottomSheetBehavior.setState(BottomSheetBehavior.STATE\_COLLAPSED);
63. Toast.makeText(**this**, "登陆成功", Toast.LENGTH\_SHORT).show();
64. }
65. dialog.dismiss();
66. }
67. });
68. });
69. dialog.show();
70. dialog.getButton(DialogInterface.BUTTON\_NEGATIVE).setTextColor(Color.BLACK);
71. **return** **true**;
72. }
73. **return** **super**.onOptionsItemSelected(item);
74. }
76. //滑动事件
77. **private** **class** GestureListener **extends** GestureDetector.SimpleOnGestureListener {
79. **private** **final** **int** SWIPE\_MIN\_DISTANCE = 400;
80. **private** **final** **int** SWIPE\_THRESHOLD\_VELOCITY = 100;
82. @Override
83. **public** **boolean** onFling(MotionEvent e1, MotionEvent e2, **float** velocityX, **float** velocityY) {
84. **if**(e1.getX() - e2.getX() > SWIPE\_MIN\_DISTANCE && Math.abs(velocityX) >  SWIPE\_THRESHOLD\_VELOCITY) {
85. // Right to left, your code here
86. System.out.println("Right to left, your code here ");
87. **return** **true**;
88. } **else** **if** (e2.getX() - e1.getX() > SWIPE\_MIN\_DISTANCE &&  Math.abs(velocityX) > SWIPE\_THRESHOLD\_VELOCITY) {
89. Dialog loginAlertDialog = **new** Dialog(MainActivity.**this**);
90. loginAlertDialog.setContentView (R.layout.activity\_main\_user);
91. /\*\*
92. \* 获取弹出框的窗口对象及参数对象以修改对话框的布局设置,
93. \* 可以直接调用getWindow(),表示获得这个Activity的Window
94. \* 对象,这样这可以以同样的方式改变这个Activity的属性.
95. \*/
96. Window dialogWindow = loginAlertDialog.getWindow();
97. WindowManager.LayoutParams lp = dialogWindow.getAttributes();
98. dialogWindow.setGravity(Gravity.LEFT | Gravity.TOP);
99. /\*\*
100. \* lp.x与lp.y表示相对于原始位置的偏移.
101. \* 当参数值包含Gravity.LEFT时,对话框出现在左边,所以lp.x就表示相对左边的偏移,负值忽略.
102. \* 当参数值包含Gravity.RIGHT时,对话框出现在右边,所以lp.x就表示相对右边的偏移,负值忽略.
103. \* 当参数值包含Gravity.TOP时,对话框出现在上边,所以lp.y就表示相对上边的偏移,负值忽略.
104. \* 当参数值包含Gravity.BOTTOM时,对话框出现在下边,所以lp.y就表示相对下边的偏移,负值忽略.
105. \* 当参数值包含Gravity.CENTER\_HORIZONTAL时
106. \* ,对话框水平居中,所以lp.x就表示在水平居中的位置移动lp.x像素,正值向右移动,负值向左移动.
107. \* 当参数值包含Gravity.CENTER\_VERTICAL时
108. \* ,对话框垂直居中,所以lp.y就表示在垂直居中的位置移动lp.y像素,正值向右移动,负值向左移动.
109. \* gravity的默认值为Gravity.CENTER,即Gravity.CENTER\_HORIZONTAL |
110. \* Gravity.CENTER\_VERTICAL.
111. \*/
112. //        lp.x = -10; // 新位置X坐标
113. //        lp.y = -10; // 新位置Y坐标
114. //        lp.width = 300; // 宽度
115. //        lp.height = 300; // 高度
116. //        lp.alpha = 0.7f; // 透明度
117. dialogWindow.setAttributes(lp);
118. /\*\*
119. \* 将对话框的大小按屏幕大小的百分比设置
120. \*/
121. WindowManager m = getWindowManager();
122. Display d = m.getDefaultDisplay(); // 获取屏幕宽、高用
123. WindowManager.LayoutParams p = dialogWindow.getAttributes(); // 获取对话框当前的参数值
124. p.height = (**int**) (d.getHeight() \* 1); // 高度设置为屏幕的1
125. p.width = (**int**) (d.getWidth() \* 0.85); // 宽度设置为屏幕的0.85
126. dialogWindow.setAttributes(p);
127. loginAlertDialog.show();
129. //        AlertDialog.Builder loginAlertDialog = new AlertDialog.Builder (MainActivity.this);
130. //        loginAlertDialog.setView (R.layout.activity\_main\_user);
131. //        loginAlertDialog.show ();
132. System.out.println("Left to right, your code here \n ");
133. **return** **true**;
134. }
135. **return** **false**;
136. }
137. }
138. }

**SearchActivity**

1. **public** **class** SearchActivity **extends** BaseActivity {
3. **private** AsyncTask searchTask = **null**;
5. @Override
6. **protected** **void** onCreate(Bundle savedInstanceState) {
7. **super**.onCreate(savedInstanceState);
9. Toolbar toolbar = findViewById(R.id.toolbar);
10. toolbar.setTitle(path);
12. setSupportActionBar(toolbar);
13. //添加默认的返回图标
14. getSupportActionBar().setDisplayHomeAsUpEnabled(**true**);
15. //设置返回键可用
16. getSupportActionBar().setHomeButtonEnabled(**true**);
17. EditText searchInput = findViewById(R.id.search\_input);
18. searchInput.setVisibility(View.VISIBLE);
20. searchInput.setOnEditorActionListener((textView, i, keyEvent) -> {
21. **if** (i == EditorInfo.IME\_ACTION\_SEARCH) {
22. search();
23. }
24. **return** **true**;
25. });
27. FloatingActionMenu fab = findViewById(R.id.fab);
28. fab.setVisibility(View.GONE);
29. }
31. @Override
32. **public** **void** init() {
33. **super**.init();
35. RecyclerView recyclerView = findViewById(R.id.file\_container);
36. adapter = **new** FileViewAdapter(**new** ArrayList<>());
37. recyclerView.setAdapter(adapter);
38. }
40. @Override
41. **public** **boolean** onOptionsItemSelected(MenuItem item) {
42. **if** (item.getItemId() == R.id.search) {
43. search();
44. **return** **true**;
45. }
46. **return** **super**.onOptionsItemSelected(item);
47. }
49. @Override
50. **public** **void** onBackPressed() {
51. **super**.onBackPressed();
52. }
54. @Override
55. **protected** **void** onDestroy() {
56. //  返回上一页前结束搜索任务
57. **if** (searchTask != **null**) {
58. searchTask.cancel(**true**);
59. }
60. **super**.onDestroy();
61. }
63. //  搜索文件
64. **public** **void** search() {
65. // 关闭搜索软键盘
66. InputMethodManager imm = (InputMethodManager) getSystemService(Context.INPUT\_METHOD\_SERVICE);
67. **if** (imm != **null**) {
68. imm.hideSoftInputFromWindow(getWindow().getDecorView().getWindowToken(), 0);
69. }
70. // 检测搜索文本是否为空
71. EditText searchInput = findViewById(R.id.search\_input);
72. **if** (StringUtils.isBlank(searchInput.getText())) {
73. **return**;
74. }
75. // 清除原有的搜索记录
76. adapter.clearFiles();
77. ProgressBar progressBar = findViewById(R.id.progress);
78. searchTask = **new** SearchTask(**this**);
80. **try** {
81. //  启动搜索异步任务
82. progressBar.setVisibility(View.VISIBLE);
83. searchTask.execute(path, searchInput.getText());
84. } **catch** (Exception e) {
85. Log.i("search", "search error", e);
86. Toast.makeText(**this**, "搜索失败:" + e.getMessage(), Toast.LENGTH\_SHORT);
87. }
88. }
90. //  自定义异步任务
91. **static** **class** SearchTask **extends** AsyncTask {
92. **private** **final** ProgressBar progressBar;
93. **private** **final** BaseActivity activity;
95. **public** SearchTask(BaseActivity activity) {
96. **this**.activity = activity;
97. progressBar = activity.findViewById(R.id.progress);
98. }
100. @Override
101. **protected** **void** onPreExecute() {
102. progressBar.setVisibility(View.VISIBLE);
103. }
105. @Override
106. **protected** Object doInBackground(Object[] objects) {
107. //  后台静默搜索，搜索到的结果通过 UI 线程添加到视图上
108. FileManagerUtils.Instance.searchFiles(**this**, **new** File(objects[0].toString()), objects[1].toString(), file -> {
109. activity.runOnUiThread(() -> activity.adapter.addFile(file));
110. });
111. **return** **null**;
112. }
114. @Override
115. **protected** **void** onPostExecute(Object o) {
116. progressBar.setVisibility(View.GONE);
117. **super**.onPostExecute(o);
118. }
120. @Override
121. **protected** **void** onCancelled() {
122. progressBar.setVisibility(View.GONE);
123. **super**.onCancelled();
124. }
125. }
126. }