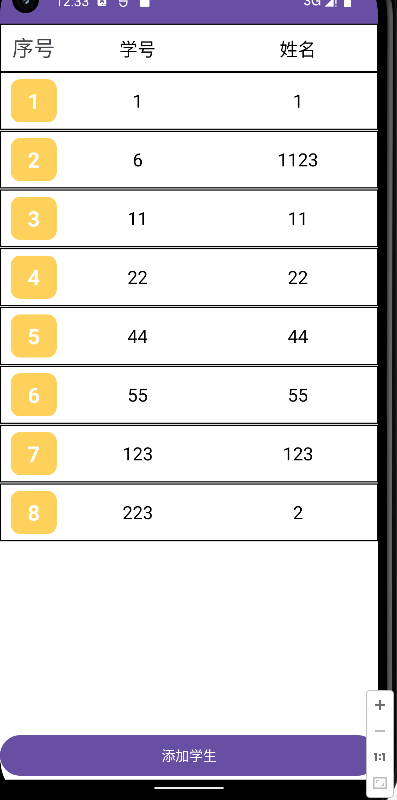
# 项目总结报告

项目名称： 学生信息管理App

学号姓名：

功能设计及运行结果：

进入应用展示学生信息的列表页面，点击列表上的某一项进入学生信息详情页，点击列表页中的按钮可以进入新增学生信息页，学生信息详情页可输入及查看学生详细信息，学生信息保存在Room数据库中，App重启后数据不丢失，并使用Fragment实现页面，可以从详情页返回列表页。

** **

学生列表 添加学生

**核心代码：**

这段代码是学生信息界面的实现，通过使用Fragment来管理不同页面的显示。在MainActivity类中定义了几个静态变量和成员变量，包括了fragmentManager、Add和List还有可能的学生对象students。在onCreate方法中，首先调用了父类的onCreate方法，并通过调用setContentView方法设置了布局文件activity\_main作为主界面的内容。然后获取FragmentManager的实例，并调用setChioceItem方法来初始化显示第一个选项卡。setChioceItem方法根据传入的参数index选择显示不同的Fragment。在该方法中，首先创建一个FragmentTransaction对象，然后通过hideFragments方法隐藏其他的Fragment。接着，根据传入的index参数，判断要显示哪个Fragment，并执行相应的操作。如果对应的Fragment对象为null，则创建一个新的Fragment对象并添加到布局中；如果对象已存在，则显示该Fragment并调用其onResume方法。hideFragments方法用于隐藏所有的Fragment。如果one或two对象不为null，则隐藏对应的Fragment。setview方法用于在其他地方调用来切换Fragment的显示。具体来说，该方法会调用setChioceItem方法，传入对应的参数来切换显示不同的页面。

|  |
| --- |
| @Override protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.*activity\_main*);  *fragmentManager* = getSupportFragmentManager();   *setChioceItem*(0); *// 初始化页面加载时显示第一个选项卡* }  @SuppressLint("ResourceAsColor") private static void setChioceItem(final int index) {  FragmentTransaction fragmentTransaction = *fragmentManager*.beginTransaction();  *hideFragments*(fragmentTransaction);  switch (index) {  case 0:  if (*list* == null) {  *list* = new List();  fragmentTransaction.add(R.id.*content*, *list*);  }  else {  fragmentTransaction.show(*list*);  *list*.onResume();  }   break;  case 1:  if (*add* == null) {  *add* = new Add();  fragmentTransaction.add(R.id.*content*, *add*);  } else {  fragmentTransaction.show(*add*);  *add*.onResume();  }  break;  }  fragmentTransaction.commit(); } private static void hideFragments(FragmentTransaction fragmentTransaction) {  if (*list* != null) {  fragmentTransaction.hide(*list*);  }  if (*add* != null) {  fragmentTransaction.hide(*add*);  }  } public static void setview(int n){  *setChioceItem*(n); } |

这段代码实现了一个使用Room库创建和管理SQLite数据库的类MyDatabase。@Database注解用于标记这个类是一个数据库类，并提供了一些参数来配置数据库。entities参数指定了数据库中的实体类，这里只有一个实体类Students。version参数表示数据库的版本号，exportSchema参数表示是否导出数据库模式。MyDatabase类继承自RoomDatabase类，以便使用Room库的功能来创建和管理数据库。studentDao用于获取StudentDao对象，通过StudentDao可以进行对Students表的增删改查等操作。在MyDatabase类中定义了一个私有常量DATABASE\_NAME，用于指定数据库的名称。databaseInstance用于存储MyDatabase类的唯一实例。getInstance方法是一个静态同步方法，用于获取MyDatabase类的实例。如果databaseInstance为空，则创建一个新的数据库实例，使用Room.databaseBuilder方法来构建数据库。其中，context.getApplicationContext()用于获取应用的全局上下文，MyDatabase.class表示要构建的数据库类，DATABASE\_NAME表示数据库的名称。最后调用build方法来构建数据库实例。如果databaseInstance不为空，则直接返回已存在的数据库实例。

|  |
| --- |
| @Database(entities = {Students.class},version = 1, exportSchema = false) public abstract class MyDatabase extends RoomDatabase {  public abstract StudentDao studentDao();   *//定义数据库名称* private static final String *DATABASE\_NAME* = "my\_db";   private static MyDatabase *databaseInstance*;   public static synchronized MyDatabase getInstance(Context context){  if(*databaseInstance* == null){  *databaseInstance* = Room.*databaseBuilder*(  context.getApplicationContext(),  MyDatabase.class,  *DATABASE\_NAME*)  .build();  }  return *databaseInstance*;  } } |

StudentDao接口定义了对学生信息进行数据库操作的方法。insertStudent()方法用于向数据库中插入学生信息。deleteStudent()方法用于删除指定的学生信息。updateStudent()方法用于更新指定学生的信息。getStudentList()方法用于查询数据库中的所有学生信息，并以列表形式返回结果。getStudentById()方法用于根据学生的 ID 查询数据库中的一条学生记录，并返回该学生的信息。

|  |
| --- |
| @Dao public interface StudentDao {   *//添加学生信息* @Insert  void insertStudent(Students student);  *//修改学生* @Update  void updateStudent(Students student);   *//查询所有学生* @Query("SELECT \* FROM students")  List<Students> getStudentList();   *//查询某个学生* @Query("SELECT \* FROM students WHERE id = :id")  Students getStudentById(int id); |

onCreateView通过inflater.inflate(R.layout.one, container, false)将布局文件R.layout.one转换为View对象，并赋值给成员变量view。view.findViewById(R.id.add)找到布局中id为add的View，并设置点击事件监听器，当点击时调用MainActivity的setview方法并传入参数 1。最后调用setListView()方法来设置ListView的数据源和适配器，并返回该 View。setListView方法通过 view.findViewById(R.id.listView)找到布局中的ListView，然后，创建 MyDatabase实例，并在新线程中执行数据库操作。在数据库操作中，首先获取学生列表，并将其转化为适合SimpleAdapter使用的数据格式，接着，在UI线程中使用 SimpleAdapter设置ListView的适配器，并为ListView的每个项设置点击事件监听器，当点击时将 MainActivity 中的学生信息更新为所点击项对应的学生，并调用 MainActivity的setview方法并传入参数1。

|  |
| --- |
| @Override public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {  view = inflater.inflate(R.layout.*one*, container, false);  Button add = view.findViewById(R.id.*add*);  add.setOnClickListener(v -> {  MainActivity.*sign*=0;  MainActivity.*setview*(1);  });  setListView();  return view; }  private void setListView() {  listView = view.findViewById(R.id.*listView*);  MyDatabase myDatabase = MyDatabase.*getInstance*(getContext());  new Thread(() -> {  list = myDatabase.studentDao().getStudentList();  if (list != null) {  data = new ArrayList<>();  for (int i = 0; i < list.size(); i++) {  Map<String, Object> map = new HashMap<>();  map.put("sum", i + 1);  map.put("id", list.get(i).id);  map.put("name", list.get(i).name);  data.add(map);  }   getActivity().runOnUiThread(() -> {  SimpleAdapter simpleAdapter = new SimpleAdapter(getContext(),  data,  R.layout.*item*,  new String[]{"sum", "id", "name"},  new int[]{R.id.*item\_sum*, R.id.*item\_id*, R.id.*item\_name*});  listView.setAdapter(simpleAdapter);  listView.setOnItemClickListener((parent, view, position, id) -> {  MainActivity.*students* = list.get(position);  MainActivity.*sign* = 1;  MainActivity.*setview*(1);  });  });  }  }).start(); } |

实现思路及心得：

创建一个数据库用于存储学生数据，用一个学生类表示学生对象，包含学生的信息，再创建一个StudentDao类用于访问数据库，提供增删改查等操作；StudentRepository类用于管理StudentDao和提供业务逻辑，如获取学生列表、添加学生、删除学生等操作；再使用Fragment进行用户交互，提供获取学生列表、添加学生、删除学生等方法。这学期学习了安卓开发Roo数据库，了解了android的基础知识，以及掌握了java语言。并且通过这个大作业还深入了解并应用了Room数据库。本应用不足的是没有提供登陆功能，也没有头像长传功能，可以优化一个登录和注册功能，并记录在数据库中保存登录信息；再提供一个头像上传功能。