6.1“登录”界面内功能的实现

代码是一个 Android 应用中的登录页面实现，它主要负责用户的登录功能。代码通过 EditText 获取用户输入的用户名和密码，并使用 Button 触发登录操作。当用户点击登录按钮时，应用会调用 UserDAO 类的 loginUser() 方法验证用户名和密码的正确性，如果验证成功，则显示“登录成功”的提示并跳转到主界面 MainActivity；如果验证失败，则显示“用户名或密码错误”的提示。

此外，代码还包含一个用于注册的新用户的 TextView。当用户点击这个 TextView 时，应用会跳转到注册页面 SigninActivity。UserDAO 类在 onCreate() 方法中打开数据库连接，在 onDestroy() 方法中关闭连接，以确保数据库资源的正确管理。

|  |
| --- |
| public class LoginActivity extends AppCompatActivity {  private EditText usernameEditText, passwordEditText;  private Button loginButton;  private TextView textviewnRegister;  private UserDAO userDAO;  @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_login);  usernameEditText = findViewById(R.id.et\_account);  passwordEditText = findViewById(R.id.et\_password);  loginButton = findViewById(R.id.btn\_Login);  textviewnRegister = findViewById(R.id.tv\_Sign);  userDAO = new UserDAO(this);  userDAO.open();  loginButton.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  String username = usernameEditText.getText().toString();  String password = passwordEditText.getText().toString();  if (userDAO.loginUser(username, password)) {  Toast.makeText(LoginActivity.this, "登录成功！", Toast.LENGTH\_SHORT).show();  Intent intent=new Intent(LoginActivity.this,MainActivity.class);  startActivity(intent);  } else {  Toast.makeText(LoginActivity.this, "用户名或密码错误！", Toast.LENGTH\_SHORT).show();  }  }  });  textviewnRegister.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  Intent intent=new Intent(LoginActivity.this, SigninActivity.class);  startActivity(intent);  }  });  }  @Override  protected void onDestroy() {  super.onDestroy();  userDAO.close();  }  } |

private SQLiteDatabase database; 和 private DatabaseHelper dbHelper;：声明了一个 SQLiteDatabase 对象和一个 DatabaseHelper 对象，用于执行数据库操作和管理数据库的创建和版本控制。

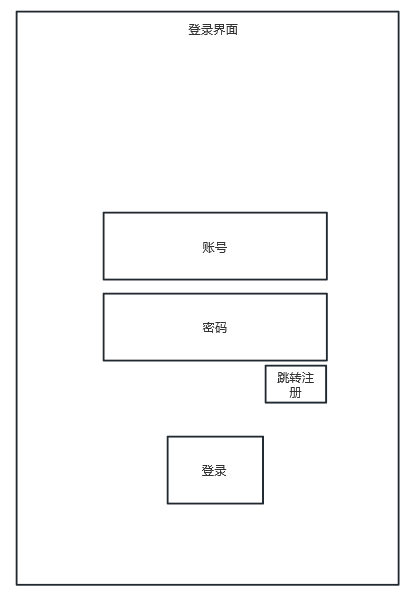
public UserDAO(Context context)：构造方法，接受一个 Context 对象作为参数，并在内部实例化了一个 DatabaseHelper 对象。

public void open() 和 public void close()：分别用于打开和关闭数据库连接。在 open() 方法中获取可写的数据库实例，在 close() 方法中关闭数据库连接。

public long addUser(User user)：用于向用户表中插入新的用户信息。将用户名和密码包装到 ContentValues 对象中，然后调用 database.insert() 方法将数据插入到名为 TABLE\_USERS 的表中。

public boolean loginUser(String username, String password)：用于验证用户登录。通过查询数据库表 TABLE\_USERS 来检查输入的用户名和密码是否匹配。如果查询结果集中有数据行，则表示用户名和密码匹配，返回 true；否则返回 false。

|  |
| --- |
| public class UserDAO {  private SQLiteDatabase database;  private DatabaseHelper dbHelper;  public UserDAO(Context context) {  dbHelper = new DatabaseHelper(context);  }  public void open() {  database = dbHelper.getWritableDatabase();  }  public void close() {  dbHelper.close();  }  public long addUser(User user) {  ContentValues values = new ContentValues();  values.put(DatabaseHelper.COLUMN\_USERNAME, user.getUsername());  values.put(DatabaseHelper.COLUMN\_PASSWORD, user.getPassword());  return database.insert(DatabaseHelper.TABLE\_USERS, null, values);  }  public boolean loginUser(String username, String password) {  String[] columns = {  DatabaseHelper.COLUMN\_ID  };  String selection = DatabaseHelper.COLUMN\_USERNAME + " = ?" + " AND " + DatabaseHelper.COLUMN\_PASSWORD + " = ?";  String[] selectionArgs = {username, password};  Cursor cursor = database.query(DatabaseHelper.TABLE\_USERS,  columns,  selection,  selectionArgs,  null,  null,  null);  int count = cursor.getCount();  cursor.close();  return count > 0;  }  } |



6.2“注册”界面内功能的实现

private EditText usernameEditText, passwordEditText;：声明了两个 EditText 对象，用于输入用户名和密码。

private UserDAO userDAO;：声明了一个 UserDAO 对象，用于执行与用户数据相关的数据库操作。

onCreate(Bundle savedInstanceState) 方法：在该方法中进行界面初始化和事件监听的设置。首先调用 setContentView() 方法设置界面布局，然后通过 findViewById() 方法找到界面上的用户名和密码输入框，并实例化 UserDAO 对象并调用其 open() 方法打开数据库连接。

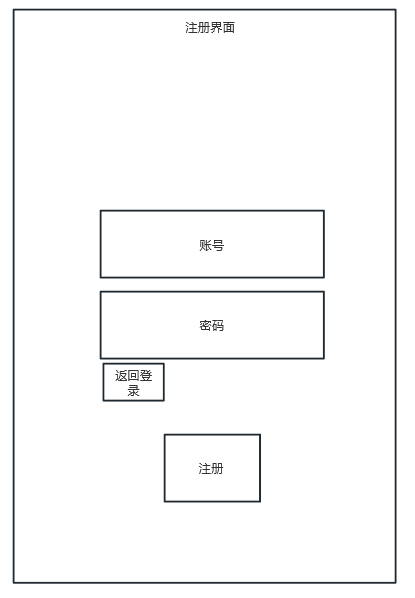
通过 findViewById().setOnClickListener() 方法分别设置了 "返回" 按钮和 "注册" 按钮的点击事件监听器，当点击这两个按钮时会执行相应的动作。

点击 "返回" 按钮时，通过 finish() 方法关闭当前界面；点击 "注册" 按钮时，将获取输入的用户名和密码，然后调用 UserDAO 的 addUser() 方法来添加新用户。根据添加用户的返回结果，显示相应的提示信息。

onDestroy() 方法：在该方法中调用 UserDAO 的 close() 方法关闭数据库连接。

总体来说，SigninActivity 类是一个用于用户注册界面的活动，其中包含了界面初始化、事件监听和数据库操作等逻辑。通过封装数据库操作到 UserDAO 类中，实现了逻辑的分离和模块化，提高了代码的可维护性和可读性。

|  |
| --- |
| public class SigninActivity extends AppCompatActivity {  private EditText usernameEditText, passwordEditText;  private UserDAO userDAO;  @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_signin);  usernameEditText = findViewById(R.id.et\_account);  passwordEditText = findViewById(R.id.et\_password);  userDAO = new UserDAO(this);  userDAO.open();  findViewById(R.id.tv\_Back).setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  finish();  }  });  findViewById(R.id.btn\_Sign).setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  String username = usernameEditText.getText().toString();  String password = passwordEditText.getText().toString();  long result = userDAO.addUser(new User(0, username, password));  if (result != -1) {  Toast.makeText(SigninActivity.this, "注册成功！", Toast.LENGTH\_SHORT).show();  } else {  Toast.makeText(SigninActivity.this, "注册失败！", Toast.LENGTH\_SHORT).show();  }  }  });  }  @Override  protected void onDestroy() {  super.onDestroy();  userDAO.close();  }  } |



6.3“笔记主页”界面内功能的实现

用于展示笔记列表和对笔记进行操作的界面。在 onCreate() 方法中进行了界面初始化，包括设置布局、初始化视图、初始化数据和设置事件监听。通过 initEvent() 方法设置了按钮的点击事件监听，其中包括一个日期选择器和清空按钮的逻辑。

在 onResume() 方法中使用了 refreshDataFromDb() 方法从数据库中刷新数据，并根据当前的布局模式设置列表的显示样式。

setListLayout() 方法用于根据保存的布局模式设置列表的显示方式，可以是线性布局或者网格布局。

refreshDataFromDb() 方法用于从数据库中获取数据并刷新界面显示。

initEvent() 方法中设置了两个按钮的点击事件监听，分别是日期选择按钮和清空按钮的逻辑。在日期选择按钮的点击事件中，通过日期选择弹窗选择日期，并根据选定的日期从数据库中查询对应的笔记，并更新界面显示。

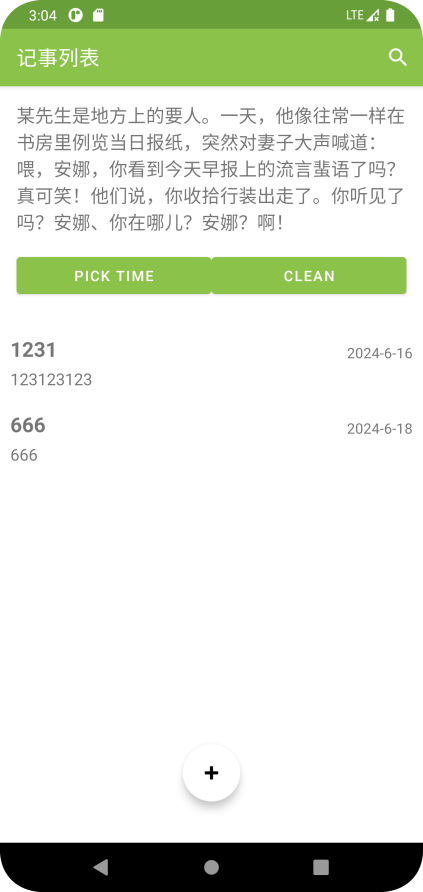
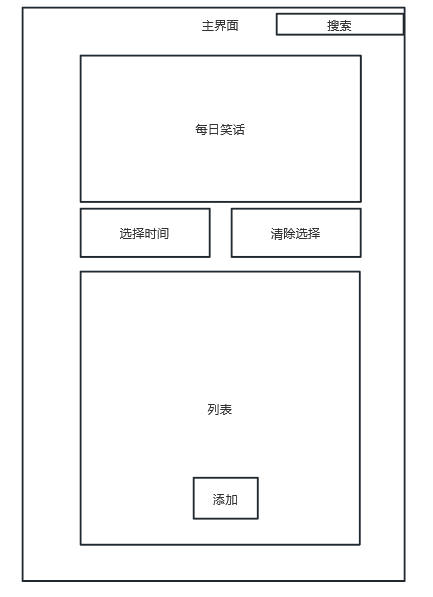
总体来说，这段代码实现了一个简单的笔记管理界面，包括了界面初始化、数据刷新、布局设置以及按钮点击事件的处理逻辑。

|  |
| --- |
| public class MainActivity extends AppCompatActivity {  private RecyclerView mRecyclerView;  private FloatingActionButton mBtnAdd;  private List<Note> mNotes;  private MyAdapter mMyAdapter;  private NoteDbOpenHelper mNoteDbOpenHelper;  public static final int MODE\_LINEAR = 0;  public static final int MODE\_GRID = 1;  public static final String KEY\_LAYOUT\_MODE = "key\_layout\_mode";  private int currentListLayoutMode = MODE\_LINEAR;  // EditText search\_box;  Button time\_picker\_button;  @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_main);  initView();  initData();  initEvent();  }  @Override  protected void onResume() {  super.onResume();  refreshDataFromDb();  setListLayout();  }  private void setListLayout() {  currentListLayoutMode = SpfUtil.getIntWithDefault(this, KEY\_LAYOUT\_MODE, MODE\_LINEAR);  if (currentListLayoutMode == MODE\_LINEAR) {  setToLinearList();  }else{  setToGridList();  }  }  private void refreshDataFromDb() {  mNotes = getDataFromDB();  mMyAdapter.refreshData(mNotes);  }  private void initEvent() {  mMyAdapter = new MyAdapter(this, mNotes);  mRecyclerView.setAdapter(mMyAdapter);  time\_picker\_button.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  // 获取当前日期  Calendar calendar = Calendar.getInstance();  int year = calendar.get(Calendar.YEAR);  int month = calendar.get(Calendar.MONTH);  int dayOfMonth = calendar.get(Calendar.DAY\_OF\_MONTH);  // 创建日期选择弹窗  DatePickerDialog datePickerDialog = new DatePickerDialog(MainActivity.this,  new DatePickerDialog.OnDateSetListener() {  @Override  public void onDateSet(DatePicker view, int selectedYear, int selectedMonth, int selectedDayOfMonth) {  // 将选定的日期设置为按钮文本  String selectedDate = selectedYear + "-" + (selectedMonth + 1) + "-" + selectedDayOfMonth;  time\_picker\_button.setText("选择的日期:"+selectedDate);  mNotes=mNoteDbOpenHelper.queryFromDbByDate(time\_picker\_button.getText().toString());  mMyAdapter.setmBeanList(mNotes);  mMyAdapter.notifyDataSetChanged();  }  }, year, month, dayOfMonth);  // 显示日期选择弹窗  datePickerDialog.show();  }  });  findViewById(R.id.clean\_button).setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  refreshDataFromDb();  time\_picker\_button.setText("Pick Time");  }  });  } |

|  |
| --- |
| public class MyAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder> {  private List<Note> mBeanList;  private LayoutInflater mLayoutInflater;  private Context mContext;  private NoteDbOpenHelper mNoteDbOpenHelper;  private int viewType;  public static int TYPE\_LINEAR\_LAYOUT = 0;  public static int TYPE\_GRID\_LAYOUT = 1;  public List<Note> getmBeanList() {  return mBeanList;  }  public void setmBeanList(List<Note> mBeanList) {  this.mBeanList = mBeanList;  }  public MyAdapter(Context context, List<Note> mBeanList){  this.mBeanList = mBeanList;  this.mContext = context;  mLayoutInflater = LayoutInflater.from(mContext);  mNoteDbOpenHelper = new NoteDbOpenHelper(mContext);  }  public void setViewType(int viewType) {  this.viewType = viewType;  }  @Override  public int getItemViewType(int position) {  return viewType;  }  @NonNull  @Override  public RecyclerView.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {  if(viewType == TYPE\_LINEAR\_LAYOUT){  View view = mLayoutInflater.inflate(R.layout.list\_item\_layout, parent, false);  MyViewHolder myViewHolder = new MyViewHolder(view);  return myViewHolder;  }else if(viewType == TYPE\_GRID\_LAYOUT){  View view = mLayoutInflater.inflate(R.layout.list\_item\_grid\_layout, parent, false);  MyGridViewHolder myGridViewHolder = new MyGridViewHolder(view);  return myGridViewHolder;  }  return null;  }  @Override  public void onBindViewHolder(@NonNull RecyclerView.ViewHolder holder, int position) {  if (holder == null) {  return;  }  if(holder instanceof MyViewHolder){  bindMyViewHolder((MyViewHolder) holder, position);  } else if (holder instanceof MyGridViewHolder) {  bindGridViewHolder((MyGridViewHolder) holder, position);  }  }  private void bindMyViewHolder(MyViewHolder holder, int position) {  Note note = mBeanList.get(position);  holder.mTvTitle.setText(note.getTitle());  holder.mTvContent.setText(note.getContent());  holder.mTvTime.setText(note.getCreatedTime());  holder.rlContainer.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  Intent intent = new Intent(mContext, EditActivity.class);  intent.putExtra("note", note);  mContext.startActivity(intent);  }  });  holder.rlContainer.setOnLongClickListener(new View.OnLongClickListener() {  @Override  public boolean onLongClick(View v) {  // 弹出弹窗  Dialog dialog = new Dialog(mContext, android.R.style.ThemeOverlay\_Material\_Dialog\_Alert);  View dialogView = mLayoutInflater.inflate(R.layout.list\_item\_dialog\_layout, null);  TextView tvDelete = dialogView.findViewById(R.id.tv\_delete);  TextView tvEdit = dialogView.findViewById(R.id.tv\_edit);  tvDelete.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  int row = mNoteDbOpenHelper.deleteFromDbById(note.getId());  if (row > 0) {  removeData(position);  }  dialog.dismiss();  }  });  tvEdit.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  Intent intent = new Intent(mContext, EditActivity.class);  intent.putExtra("note", note);  mContext.startActivity(intent);  dialog.dismiss();  }  });  dialog.setContentView(dialogView);  dialog.setCanceledOnTouchOutside(true);  dialog.show();  return true;  }  });  }  private void bindGridViewHolder(MyGridViewHolder holder, int position) {  Note note = mBeanList.get(position);  holder.mTvTitle.setText(note.getTitle());  holder.mTvContent.setText(note.getContent());  holder.mTvTime.setText(note.getCreatedTime());  holder.rlContainer.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  Intent intent = new Intent(mContext, EditActivity.class);  intent.putExtra("note", note);  mContext.startActivity(intent);  }  });  holder.rlContainer.setOnLongClickListener(new View.OnLongClickListener() {  @Override  public boolean onLongClick(View v) {  // 弹出弹窗  Dialog dialog = new Dialog(mContext, android.R.style.ThemeOverlay\_Material\_Dialog\_Alert);  View dialogView = mLayoutInflater.inflate(R.layout.list\_item\_dialog\_layout, null);  TextView tvDelete = dialogView.findViewById(R.id.tv\_delete);  TextView tvEdit = dialogView.findViewById(R.id.tv\_edit);  tvDelete.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  int row = mNoteDbOpenHelper.deleteFromDbById(note.getId());  if (row > 0) {  removeData(position);  }  dialog.dismiss();  }  });  tvEdit.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  Intent intent = new Intent(mContext, EditActivity.class);  intent.putExtra("note", note);  mContext.startActivity(intent);  dialog.dismiss();  }  });  dialog.setContentView(dialogView);  dialog.setCanceledOnTouchOutside(true);  dialog.show();  return true;  }  });  }  @Override  public int getItemCount() {  return mBeanList.size();  }  public void refreshData(List<Note> notes) {  this.mBeanList = notes;  notifyDataSetChanged();  }  public void removeData(int pos) {  mBeanList.remove(pos);  notifyItemRemoved(pos);  }  class MyViewHolder extends RecyclerView.ViewHolder{  TextView mTvTitle;  TextView mTvContent;  TextView mTvTime;  ViewGroup rlContainer;  public MyViewHolder(@NonNull View itemView) {  super(itemView);  this.mTvTitle = itemView.findViewById(R.id.tv\_title);  this.mTvContent = itemView.findViewById(R.id.tv\_content);  this.mTvTime = itemView.findViewById(R.id.tv\_time);  this.rlContainer = itemView.findViewById(R.id.rl\_item\_container);  }  }  class MyGridViewHolder extends RecyclerView.ViewHolder{  TextView mTvTitle;  TextView mTvContent;  TextView mTvTime;  ViewGroup rlContainer;  public MyGridViewHolder(@NonNull View itemView) {  super(itemView);  this.mTvTitle = itemView.findViewById(R.id.tv\_title);  this.mTvContent = itemView.findViewById(R.id.tv\_content);  this.mTvTime = itemView.findViewById(R.id.tv\_time);  this.rlContainer = itemView.findViewById(R.id.rl\_item\_container);  }  }  } |

笑话API

|  |
| --- |
| 用于从一个特定的API获取笑话并解析JSON响应。它分为几个部分：  导入包: 首先，我们导入了一些必要的Java类库，以便能够进行HTTP请求和处理JSON数据。  JokeFetcher类定义: 这个类包含了一些常量、变量和方法。  在类中，有一个常量API\_URL，它包含了要请求的API的URL。  有一个OkHttpClient实例和一个Gson实例，它们分别用来发送HTTP请求和解析JSON数据。  类中定义了一个fetchJoke方法，用于发送HTTP请求以获取笑话数据。它使用OkHttpClient来创建并执行请求。  JokeResponse内部类: 这是一个嵌套的内部类，用于对应API返回的JSON数据结构。它包含了一些字段，用来表示请求结果的原因或状态信息，以及笑话数据的具体内容。  总的来说，这段代码实现了通过HTTP请求获取笑话数据，并且利用Gson库解析JSON响应。这样，我们就可以轻松地在Java应用程序中使用这些笑话数据，比如显示在用户界面上或者进行其他操作。 |
| import okhttp3.Call;  import okhttp3.Callback;  import okhttp3.OkHttpClient;  import okhttp3.Request;  import okhttp3.Response;  import com.google.gson.Gson;  import java.io.IOException;  public class JokeFetcher {  private static final String API\_URL = "http://v.juhe.cn/joke/content/list.php?sort=&page=&pagesize=1&time=1418816972&key=3f3c7828\*\*\*\*\*\*1f18f897";  private OkHttpClient client;  private Gson gson;  public JokeFetcher() {  client = new OkHttpClient();  gson = new Gson();  }  public void fetchJoke(Callback callback) {  Request request = new Request.Builder()  .url(API\_URL)  .build();  client.newCall(request).enqueue(callback);  }  public static class JokeResponse {  String reason;  Result result;  int error\_code;  public static class Result {  Data[] data;  public static class Data {  String content;  String hashId;  long unixtime;  String updatetime;  }  }  }  } |



6.4“添加笔记”界面内功能的实现

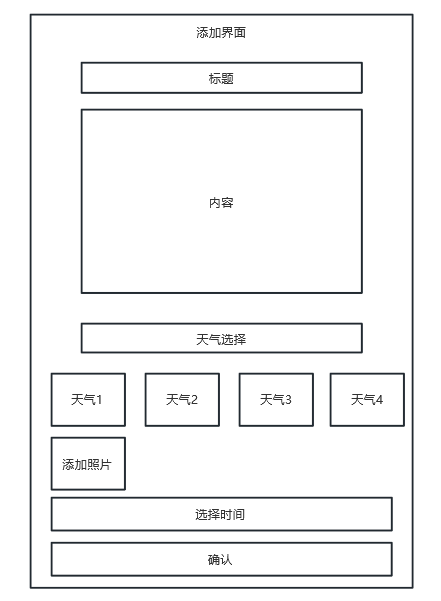
AddActivity 的类，用于添加新的笔记信息。在 onCreate() 方法中进行了界面初始化，包括设置布局、初始化视图和设置按钮的点击事件监听。其中，通过点击图片按钮可以打开相册选择图片，并在选择完毕后将图片显示在界面上。在 add(View view) 方法中，首先获取用户输入的标题和内容，然后将用户选择的图片转换为二进制数组，并将所有信息包括天气、时间等保存到数据库中。最后根据数据库操作的结果弹出相应的提示信息。另外，还重写了 onActivityResult() 方法用于处理从相册选择图片后返回的结果，将选择的图片显示在界面上。总体来说，这段代码实现了一个笔记的添加功能，包括了界面初始化、图片选择、数据处理、数据库操作和结果提示等逻辑。

|  |
| --- |
| public class AddActivity extends AppCompatActivity {  private EditText etTitle,etContent;  private NoteDbOpenHelper mNoteDbOpenHelper;  RadioButton sunny\_radio\_button,rainy\_radio\_button,overcast\_radio\_button,cloudy\_radio\_button;  ImageButton ivPhoto;  Button btn\_Time;  @Override  protected void onCreate(Bundle savedInstanceState) {  super.onCreate(savedInstanceState);  setContentView(R.layout.activity\_add);  etTitle = findViewById(R.id.et\_title);  etContent = findViewById(R.id.et\_content);  btn\_Time = findViewById(R.id.btn\_Time);  mNoteDbOpenHelper = new NoteDbOpenHelper(this);  sunny\_radio\_button = findViewById(R.id.sunny\_radio\_button);  rainy\_radio\_button = findViewById(R.id.rainy\_radio\_button);  overcast\_radio\_button = findViewById(R.id.overcast\_radio\_button);  cloudy\_radio\_button = findViewById(R.id.cloudy\_radio\_button);  ivPhoto = findViewById(R.id.iv\_photo);  ivPhoto.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  Intent intent = new Intent(Intent.ACTION\_PICK,null);  intent.setDataAndType(MediaStore.Images.Media.EXTERNAL\_CONTENT\_URI,"image/\*");  startActivityForResult(intent,1);  }  });  btn\_Time.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  // 获取当前日期  Calendar calendar = Calendar.getInstance();  int year = calendar.get(Calendar.YEAR);  int month = calendar.get(Calendar.MONTH);  int dayOfMonth = calendar.get(Calendar.DAY\_OF\_MONTH);  // 创建日期选择弹窗  DatePickerDialog datePickerDialog = new DatePickerDialog(AddActivity.this,  new DatePickerDialog.OnDateSetListener() {  @Override  public void onDateSet(DatePicker view, int selectedYear, int selectedMonth, int selectedDayOfMonth) {  // 将选定的日期设置为按钮文本  String selectedDate = selectedYear + "-" + (selectedMonth + 1) + "-" + selectedDayOfMonth;  btn\_Time.setText(selectedDate);  }  }, year, month, dayOfMonth);  // 显示日期选择弹窗  datePickerDialog.show();  }  });  }  public void add(View view) {  String title = etTitle.getText().toString();  String content = etContent.getText().toString();  if (TextUtils.isEmpty(title)) {  ToastUtil.toastShort(this, "标题不能为空！");  return;  }  Note note = new Note();  note.setTitle(title);  note.setContent(content);  //把图片先转化成bitmap格式  BitmapDrawable drawable = (BitmapDrawable) ivPhoto.getDrawable();  Bitmap bitmap = drawable.getBitmap();  //二进制数组输出流  ByteArrayOutputStream byStream = new ByteArrayOutputStream();  //将图片压缩成质量为100的PNG格式图片  bitmap.compress(Bitmap.CompressFormat.PNG, 40, byStream);  //把输出流转换为二进制数组  byte[] byteArray = byStream.toByteArray();  note.setPicture(byteArray);  note.setCreatedTime(btn\_Time.getText().toString());  if (sunny\_radio\_button.isChecked()){  note.setWeather(0);  }else if (rainy\_radio\_button.isChecked()){  note.setWeather(1);  }else if (overcast\_radio\_button.isChecked()){  note.setWeather(2);  }else if (cloudy\_radio\_button.isChecked()){  note.setWeather(3);  }else {  note.setWeather(0);  }  long row = mNoteDbOpenHelper.insertData(note);  if (row != -1) {  ToastUtil.toastShort(this,"添加成功！");  this.finish();  }else {  ToastUtil.toastShort(this,"添加失败！");  }  }  @Override  protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {  super.onActivityResult(requestCode, resultCode, data);  if (requestCode == 1) {  //从相册返回的数据  if (data != null) {  //得到图片的全路径  Uri uri = data.getData();  ivPhoto.setImageURI(uri);  }  }  }  private String getCurrentTimeFormat() {  SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy年MM月dd HH:mm:ss");  Date date = new Date();  return simpleDateFormat.format(date);  }  } |

这段代码是一个数据库操作方法，用于向数据库中插入笔记数据。首先通过 getWritableDatabase() 方法获取可写的数据库实例，然后创建一个 ContentValues 对象用于存放要插入的数据。接着将笔记对象中的各项数据放入 ContentValues 对象中，包括标题、内容、创建时间、天气和图片数据。最后调用数据库的 insert 方法将数据插入到名为 TABLE\_NAME\_NOTE 的表中，并返回插入操作的结果。

实现思路：获取可写的数据库实例。创建一个 ContentValues 对象并将笔记对象中的数据存入。调用数据库的 insert 方法，将数据插入到指定的表中。返回插入操作的结果，即返回插入的行号或-1表示插入失败。

|  |
| --- |
| public long insertData(Note note) {  SQLiteDatabase db = getWritableDatabase();  ContentValues values = new ContentValues();  values.put(COLUMN\_TITLE, note.getTitle());  values.put(COLUMN\_CONTENT, note.getContent());  values.put(COLUMN\_CREATE\_TIME, note.getCreatedTime());  values.put(COLUMN\_WEATHER, note.getWeather());  values.put(COLUMN\_IMAGE\_DATA, note.getPicture()); // 假设getImageData()返回图片的byte[]数据  return db.insert(TABLE\_NAME\_NOTE, null, values);  } |



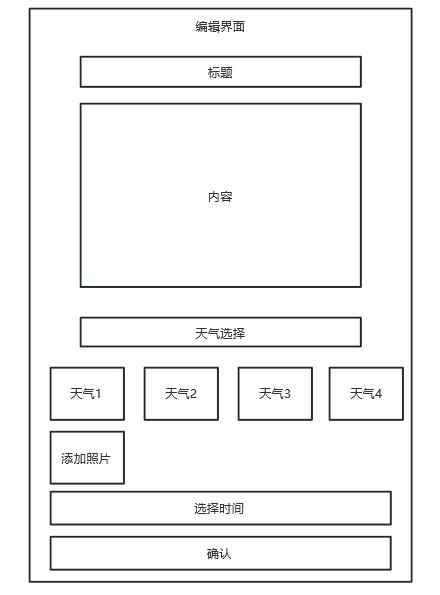
6.5“编辑笔记”界面内功能的实现

这段代码是一个数据库操作方法，用于更新数据库中的笔记数据。首先通过 getWritableDatabase() 方法获取可写的数据库实例，然后创建一个 ContentValues 对象用于存放要更新的数据。接着将笔记对象中的各项数据放入 ContentValues 对象中，包括标题、内容、创建时间、天气和图片数据（更新图片数据）。最后调用数据库的 update 方法，使用 WHERE 子句指定更新的条件，即根据笔记的 ID 进行更新，并返回更新操作的结果。

实现思路：

获取可写的数据库实例。创建一个 ContentValues 对象并将笔记对象中的数据存入。调用数据库的 update 方法，传入更新的表名、更新的数据和更新的条件。返回更新操作的结果，即返回受影响的行数。

|  |
| --- |
| public int updateData(Note note) {  SQLiteDatabase db = getWritableDatabase();  ContentValues values = new ContentValues();  values.put(COLUMN\_TITLE, note.getTitle());  values.put(COLUMN\_CONTENT, note.getContent());  values.put(COLUMN\_CREATE\_TIME, note.getCreatedTime());  values.put(COLUMN\_WEATHER, note.getWeather());  values.put(COLUMN\_IMAGE\_DATA, note.getPicture()); // 更新图片数据  return db.update(TABLE\_NAME\_NOTE, values, COLUMN\_ID + " = ?", new String[]{note.getId()});  } |



6.6“删除笔记”界面内功能的实现

这段代码是一个数据库操作方法，用于根据笔记的ID从数据库中删除对应的笔记数据。首先通过 getWritableDatabase() 方法获取可写的数据库实例，然后调用数据库的 delete 方法，传入要删除的表名、删除的条件以及条件参数，即根据笔记的 ID 进行删除，并返回删除操作的结果。

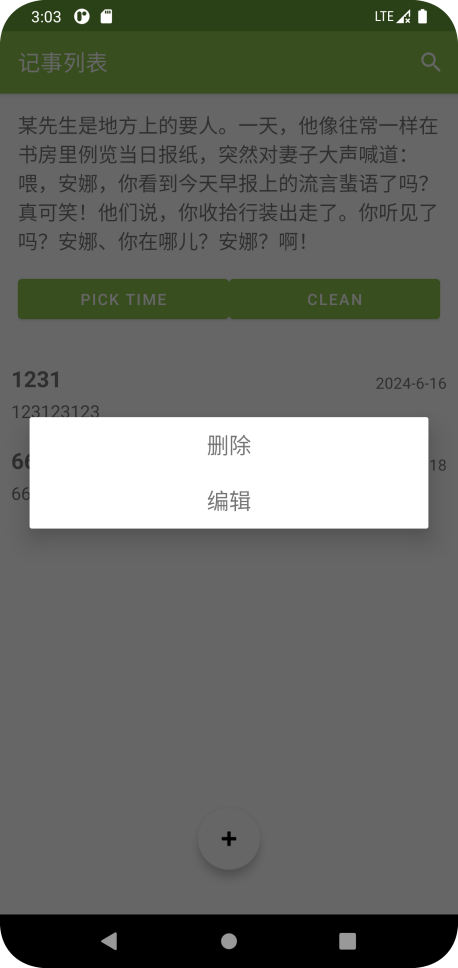
实现思路：获取可写的数据库实例。调用数据库的 delete 方法，传入要删除的表名、删除的条件和条件参数。返回删除操作的结果，即返回受影响的行数。

|  |
| --- |
| public int deleteFromDbById(String id) {  SQLiteDatabase db = getWritableDatabase();  return db.delete(TABLE\_NAME\_NOTE, COLUMN\_ID + " = ?", new String[]{id});  } |

这段代码是设置了一个长按监听器，当用户长按列表项时，会弹出一个包含删除和编辑选项的对话框。具体实现如下：

当用户长按列表项时，会触发 onLongClick 方法。在 onLongClick 方法中，创建一个 Dialog 对象，并设置其样式为 android.R.style.ThemeOverlay\_Material\_Dialog\_Alert。通过布局填充器（mLayoutInflater）加载自定义的对话框布局（R.layout.list\_item\_dialog\_layout），并在对话框中找到删除和编辑选项的视图。为删除按钮设置点击监听器，在点击时调用 mNoteDbOpenHelper.deleteFromDbById(note.getId()) 方法来删除数据库中对应的笔记数据，并根据返回的结果决定是否移除列表中的对应项。为编辑按钮设置点击监听器，在点击时创建一个带有笔记数据的 Intent 并启动编辑页面的活动（EditActivity）。设置对话框的内容视图，使其显示自定义的对话框布局。设置对话框点击外部区域时自动关闭，并显示对话框。总体来说，这段代码实现了长按列表项弹出删除和编辑选项的功能，并根据用户的操作来进行相应的数据库操作和界面更新。

|  |
| --- |
| holder.rlContainer.setOnLongClickListener(new View.OnLongClickListener() {  @Override  public boolean onLongClick(View v) {  // 弹出弹窗  Dialog dialog = new Dialog(mContext, android.R.style.ThemeOverlay\_Material\_Dialog\_Alert);  View dialogView = mLayoutInflater.inflate(R.layout.list\_item\_dialog\_layout, null);  TextView tvDelete = dialogView.findViewById(R.id.tv\_delete);  TextView tvEdit = dialogView.findViewById(R.id.tv\_edit);  tvDelete.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  int row = mNoteDbOpenHelper.deleteFromDbById(note.getId());  if (row > 0) {  removeData(position);  }  dialog.dismiss();  }  });  tvEdit.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  Intent intent = new Intent(mContext, EditActivity.class);  intent.putExtra("note", note);  mContext.startActivity(intent);  dialog.dismiss();  }  });  dialog.setContentView(dialogView);  dialog.setCanceledOnTouchOutside(true);  dialog.show();  return true;  }  }); |



6.7“搜索笔记”界面内功能的实现

这段代码是一个活动中的菜单相关方法，包括创建选项菜单和处理菜单项选择的方法。

onCreateOptionsMenu 方法用于创建选项菜单。首先通过 getMenuInflater().inflate() 方法将菜单布局文件（R.menu.menu\_main）填充到菜单对象（menu）中。然后通过 menu.findItem(R.id.menu\_search).getActionView() 获取到搜索框视图（SearchView）。

为搜索框设置一个查询文本监听器（OnQueryTextListener）。在 onQueryTextChange 方法中，根据用户输入的新文本（newText），调用 mNoteDbOpenHelper.queryFromDbByTitle(newText) 方法来查询数据库中匹配标题的笔记数据，并更新适配器（mMyAdapter）的数据源（mNotes），最后刷新界面。

onOptionsItemSelected 方法用于处理菜单项选择事件。当前代码中并没有对任何菜单项进行处理，只是将选择的菜单项设置为选中状态，然后调用父类的方法来处理其他菜单项。

总结来说，这段代码实现了在活动中创建选项菜单，并对菜单项的选择事件进行处理，其中包括监听搜索框的文本变化并根据文本内容查询数据库，并将查询结果更新到界面中。

|  |
| --- |
| @Override  public boolean onCreateOptionsMenu(Menu menu) {  getMenuInflater().inflate(R.menu.menu\_main, menu);  SearchView searchView = (SearchView) menu.findItem(R.id.menu\_search).getActionView();  searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {  @Override  public boolean onQueryTextSubmit(String query) {  return false;  }  @Override  public boolean onQueryTextChange(String newText) {  mNotes = mNoteDbOpenHelper.queryFromDbByTitle(newText);  mMyAdapter.refreshData(mNotes);  return true;  }  });  return super.onCreateOptionsMenu(menu);  }  @Override  public boolean onOptionsItemSelected(@NonNull MenuItem item) {  item.setChecked(true);  switch (item.getItemId()) {  default:  return super.onOptionsItemSelected(item);  }  } |

这段代码包含了两个点击事件的监听器，分别是时间选择按钮和清除按钮的点击事件监听。

时间选择按钮的点击事件监听器中，当用户点击按钮时会触发一个日期选择弹窗（DatePickerDialog），首先获取当前日期，并创建一个日期选择弹窗。在弹窗中，用户选择日期后会触发 onDateSet 方法，将选定的日期设置为按钮的文本，并根据选定的日期查询数据库，然后更新适配器的数据源，并通知适配器数据已经改变，以便界面刷新显示。

清除按钮的点击事件监听器中，当用户点击清除按钮时，会调用 refreshDataFromDb() 方法来刷新数据库中的数据，并将时间选择按钮的文本重置为 "Pick Time"。

总体来说，这段代码实现了时间选择按钮和清除按钮的点击事件监听，分别用于选择日期和清除筛选条件，并根据用户的操作来更新界面显示。

|  |
| --- |
| time\_picker\_button.setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  // 获取当前日期  Calendar calendar = Calendar.getInstance();  int year = calendar.get(Calendar.YEAR);  int month = calendar.get(Calendar.MONTH);  int dayOfMonth = calendar.get(Calendar.DAY\_OF\_MONTH);  // 创建日期选择弹窗  DatePickerDialog datePickerDialog = new DatePickerDialog(MainActivity.this,  new DatePickerDialog.OnDateSetListener() {  @Override  public void onDateSet(DatePicker view, int selectedYear, int selectedMonth, int selectedDayOfMonth) {  // 将选定的日期设置为按钮文本  String selectedDate = selectedYear + "-" + (selectedMonth + 1) + "-" + selectedDayOfMonth;  time\_picker\_button.setText("选择的日期:"+selectedDate);  mNotes=mNoteDbOpenHelper.queryFromDbByDate(time\_picker\_button.getText().toString());  mMyAdapter.setmBeanList(mNotes);  mMyAdapter.notifyDataSetChanged();  }  }, year, month, dayOfMonth);  // 显示日期选择弹窗  datePickerDialog.show();  }  });  findViewById(R.id.clean\_button).setOnClickListener(new View.OnClickListener() {  @Override  public void onClick(View v) {  refreshDataFromDb();  time\_picker\_button.setText("Pick Time");  }  }); |

