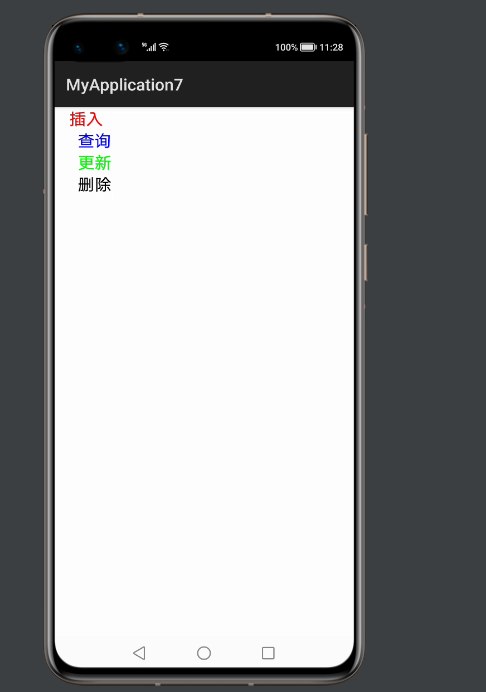
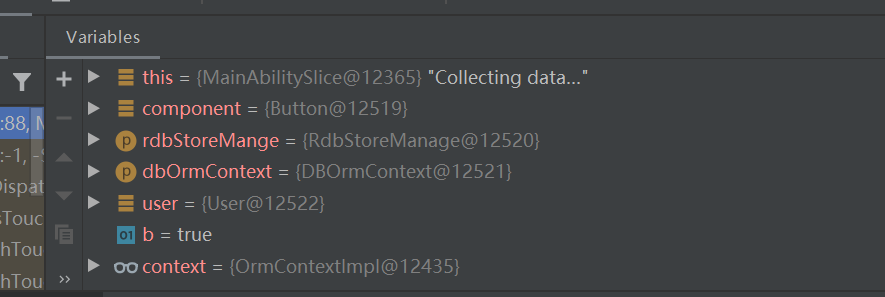
1. **ActiveOhos功能介绍**
   1. **组件介绍**

基于鸿蒙系统连接数据库进行sqlite数据库操作的时候，创建连接的时候是有些繁琐的，本组件就是为了简化sqlite的连接，并且对鸿蒙原生的API进行封装加强，使得读写sqlite数据库的时候更加方便

* 1. **phone模拟器上运行效果**



* 1. **插入数据成功**



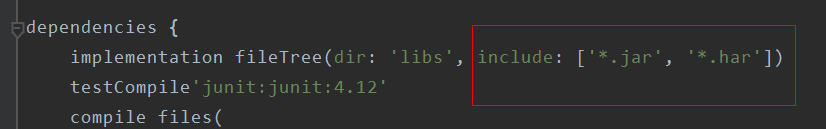
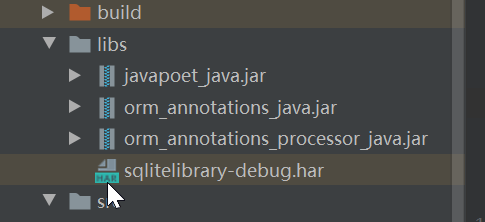
1. **ActiveOhos使用方法**
   1. **为应用添加sqlitelibrary-debug.har包依赖**

在应用模块中调用HAR，常用的添加依赖的方式包括如下两种。

* 方式一：依赖本地HAR

第一步：将**sqlitelibrary-debug.har**复制到entry\libs目录下即可（由于build.gradle中已经依赖的libs目录下的\*.har，因此不需要在做修改）。

查看工程目录中build.gradle下的\*.har是存在



第二步：除了依赖以上har之外还需要添加外部依赖用来实现类的引入，引入方式如下，引入完之后同步下就可以使用。

* 如果使用注解处理器的模块为“com.huawei.ohos.hap”模块，则需要在模块的“build.gradle”文件的“ohos”节点中添加以下配置：

compileOptions{

annotationEnabled true

}

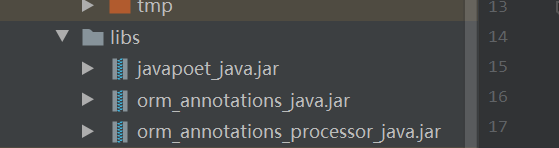
* 如果使用注解处理器的模块为“com.huawei.ohos.library”模块，则需要在模块的

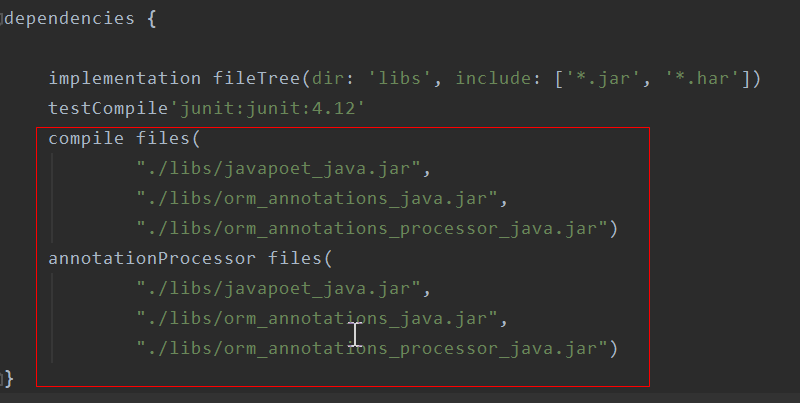
“build.gradle”文件的“dependencies”节点中配置注解处理器。查看“orm\_annotations\_java.jar”、“orm\_annotations\_processor\_java.jar” 、“javapoet\_java.jar”这3个jar包在HUAWEI SDK中的对应目录，并将目录的这三个jar包导进来。

dependencies { compile files("orm\_annotations\_java.jar的路径","orm\_annotations\_processor\_java.jar的路径","javapoet\_java.jar的路径") annotationProcessor files("orm\_annotations\_java.jar的路径","orm\_annotations\_processor\_java.jar的路径","javapoet\_java.jar的路径")}

* 如果使用注解处理器的模块为“java-library”模块，则需要在模块的“build.gradle”文件的“dependencies”节点中配置注解处理器，并导入“ohos.jar”。

dependencies { compile files("ohos.jar的路径","orm\_annotations\_java.jar的路径","orm\_annotations\_processor\_java.jar的路径","javapoet\_java.jar的路径") annotationProcessor files("orm\_annotations\_java.jar的路径","orm\_annotations\_processor\_java.jar的路径","javapoet\_java.jar的路径")}

**比如：**

以上操作无误之后就可以进行编码了！

1. **ActiveOhos开发实现**
   1. **主页面的布局文件**

定义四个按钮分别实现增删改查，定义四个Button实现请求点击事件

<?xml version="1.0" encoding="utf-8"?>  
<DirectionalLayout  
 xmlns:ohos="http://schemas.huawei.com/res/ohos"  
 ohos:height="match\_parent"  
 ohos:width="match\_parent"  
 ohos:orientation="vertical">  
  
 <Button  
 ohos:id="$+id:btn\_insert"  
 ohos:height="match\_content"  
 ohos:width="80fp"  
 ohos:text\_color="red"  
 ohos:text="插入"  
 ohos:text\_size="20fp"  
 ohos:weight="100fp"/>  
  
 <Button  
 ohos:id="$+id:btn\_query"  
 ohos:height="match\_content"  
 ohos:width="100fp"  
 ohos:text\_color="blue"  
 ohos:text="查询"  
 ohos:text\_size="20fp"  
 ohos:weight="100fp"/>  
  
 <Button  
 ohos:id="$+id:btn\_update"  
 ohos:height="match\_content"  
 ohos:width="100fp"  
 ohos:text\_color="green"  
 ohos:text="更新"  
 ohos:text\_size="20fp"  
 ohos:weight="100fp"/>  
  
 <Button  
 ohos:id="$+id:btn\_delete"  
 ohos:height="match\_content"  
 ohos:width="100fp"  
 ohos:text\_color="black"  
 ohos:text="删除"  
 ohos:text\_size="20fp"  
 ohos:weight="100fp"/>  
  
 <ListContainer  
 ohos:id="$+id:listText"  
 ohos:height="match\_parent"  
 ohos:width="match\_parent"/>  
  
</DirectionalLayout>

ohos:width="match\_content"  
 ohos:background\_element="$graphic:background\_ability\_main"  
 ohos:layout\_alignment="horizontal\_center"  
 ohos:text="get请求"  
 ohos:text\_size="50"  
 ohos:top\_margin="80vp"  
 />  
  
</DirectionalLayout>

* 1. **例子代码如下**

组件中一共有两种连接数据的方式，分别OrmContext，RdbStore ，其中OrmContext是需要定义一个实体类（User）来和数据库对应表名及字段，还有一个数据库类 BookStore 来配合开发代码如下：

***MainAbilitySlice***

import com.example.myapplication.BookStore;  
import com.example.myapplication.ResourceTable;  
import com.example.myapplication.User;  
import com.example.sqlitelibrary.DBManage;  
import com.example.sqlitelibrary.DBOrmContext;  
import com.example.sqlitelibrary.utils.Log;  
import ohos.aafwk.ability.AbilitySlice;  
import ohos.aafwk.content.Intent;  
import ohos.agp.components.Button;  
import ohos.agp.components.Component;  
import ohos.data.DatabaseHelper;  
import ohos.data.orm.OrmContext;  
import ohos.data.orm.OrmPredicates;  
import ohos.data.rdb.RdbStore;  
import ohos.data.rdb.ValuesBucket;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class MainAbilitySlice extends AbilitySlice implements Component.ClickedListener {  
   
 private DatabaseHelper helper;  
 private RdbStore store;  
 private OrmContext context;  
 @Override  
 public void onStart(Intent intent) {  
 super.onStart(intent);  
 super.setUIContent(ResourceTable.*Layout\_ability\_main*);  
 helper = new DatabaseHelper(this);  
 DBManage dbManger = new DBManage("user.db","user");  
 context = dbManger.getConnectionContext(helper, BookStore.class);  
// DBManage dbManger = new DBManage("user.db");  
// store = dbManger.getConnectionStore(helper,"user");  
 Button btnInsert = (Button) findComponentById(ResourceTable.*Id\_btn\_insert*);  
 Button btnQuery = (Button) findComponentById(ResourceTable.*Id\_btn\_query*);  
 Button btnDelete = (Button) findComponentById(ResourceTable.*Id\_btn\_delete*);  
 Button btnUpdate = (Button) findComponentById(ResourceTable.*Id\_btn\_update*);  
 btnInsert.setClickedListener(this::onClick);  
 btnQuery.setClickedListener(this::onClick);  
 btnDelete.setClickedListener(this::onClick);  
 btnUpdate.setClickedListener(this::onClick);  
 }  
  
 @Override  
 public void onActive() {  
 super.onActive();  
 }  
  
 @Override  
 public void onForeground(Intent intent) {  
 super.onForeground(intent);  
 }  
  
 @Override  
 public void onClick(Component component) {  
// RdbStoreManage rdbStoreMange = new RdbStoreManage();  
// ValuesBucket values = new ValuesBucket();  
// values.putInteger("id", 1);  
// values.putString("name", "zhangsan");  
// values.putInteger("age", 18);  
// values.putDouble("salary", 100.5);  
// values.putByteArray("blobType", new byte[] {1, 2, 3});  
// rdbStoreMange.setSql(store, "insert into user values(zhangsan, 18, 100.5, byte[1,2,3])");  
// long id = rdbStoreMange.insert(store,"user", values);  
// System.out.println(id);  
  
 DBOrmContext dbOrmContext = new DBOrmContext();  
 switch (component.getId()) {  
 case ResourceTable.*Id\_btn\_insert*: //插入数据  
 //第一次使用user对应的表的时候，如果有这张表就直接使用，没有就创建表  
 User user = new User();  
 user.setFirstName("Zhang");  
 user.setLastName("San");  
 user.setAge(29);  
 user.setBalance(100.51);  
 boolean b = dbOrmContext.insert(context, user);  
 Log.*i*("插入成功");  
 System.*out*.println(b);  
 break;  
 case ResourceTable.*Id\_btn\_query*: //条件查询  
 List<User> users = new ArrayList<>();  
 OrmPredicates query = context.where(User.class).equalTo("lastName", "San");  
 users = dbOrmContext.query(context, query);  
 break;  
 case ResourceTable.*Id\_btn\_delete*: //条件删除  
 OrmPredicates delete = context.where(User.class).equalTo("lastName", "San");  
 int delete1 = dbOrmContext.delete(context, delete);  
 System.*out*.println(delete1);  
 break;  
 case ResourceTable.*Id\_btn\_update*: //条件更新  
 ValuesBucket valuesBucket = new ValuesBucket();  
 valuesBucket.putInteger("age", 31);  
 valuesBucket.putString("firstName", "Zhang");  
 valuesBucket.putString("lastName", "San");  
 valuesBucket.putDouble("balance", 300.51);  
 OrmPredicates update = context.where(User.class).equalTo("userId", 1);  
 int update1 = dbOrmContext.update(context, valuesBucket, update);  
 System.*out*.println(update1);  
 break;  
 }  
 dbOrmContext.flush(context);  
 }  
}

***user.java***

@Entity(tableName = "user", ignoredColumns = {"ignoreColumn1", "ignoreColumn2"},  
 indices = {@Index(value = {"firstName", "lastName"}, name = "name\_index", unique = true)})  
public class User extends OrmObject {  
 // 此处将userId设为了自增的主键。注意只有在数据类型为包装类型时，自增主键才能生效。  
 @PrimaryKey(autoGenerate = true)  
 private Integer userId;  
 private String firstName;  
 private String lastName;  
 private int age;  
 private double balance;  
 private int ignoreColumn1;  
 private int ignoreColumn2;  
  
 // 开发者自行添加字段的getter和setter 方法

***BookStore.java***

@Database(entities = {User.class}, version = 1)  
public abstract class BookStore extends OrmDatabase {  
}