

****

**J I A N G S U U N I V E R S I T Y**

**“物联网移动应用开发”上机报告**

**上机四：鸿蒙本地数据存储**

|  |  |
| --- | --- |
| 学院名称： | 计算机科学与通信工程学院 |
| 专业班级： | 物联网2102 |
| 学生姓名： | 孟德昊 |
| 学生学号： | 3201303014 |
| 指导教师： | 金 华 |

**2022年 11 月 02 日**

**鸿蒙数据存储开发**

1. 实验目的
2. 了解鸿蒙本地数据库存储的相关基础知识
3. 掌握关系型数据库存储的增删改查操作
4. 掌握应用偏好数据库的相关操作
5. 掌握鸿蒙本地文件存储方法
6. 实验内容
7. 练习关系型数据库增删改查操作。
8. 练习应用偏好数据库的相关操作。
9. 练习本地文件存储方法的操作。
10. 实验详细报告
11. 实验内容1
12. 问题描述

了解鸿蒙关系型数据库存储的相关基础知识并使用鸿蒙开发中关系型数据库增删改查。

1. 设计思想与原理

关系型数据库是在SQLite基础上实现的本地数据操作机制，提供给用户无需编写原生SQL语句就能进行数据增删改查的方法，同时也支持原生SQL语句操作，关系型数据库提供了数据库创建方式，以及对应的删除接口。

1. 程序清单（含必要注释说明）

// 创建数据库

private RdbStore getTestRdbStore() {  
 *// 实例化DatabaseHelper对象* DatabaseHelper helper = new DatabaseHelper(this);  
 *// 创建可读写的test.sqlite数据库* StoreConfig config = StoreConfig.*newDefaultConfig*("test.sqlite");  
 *// 获取RdbStore对象，版本号为1* RdbStore store = helper.getRdbStore(config, 1, new RdbOpenCallback() {  
 @Override  
 public void onCreate(RdbStore rdbStore) {  
 *// 在创建test.sqlite数据库时，创建student数据表* rdbStore.executeSql("CREATE TABLE IF NOT EXISTS student (" +  
 "id INTEGER PRIMARY KEY AUTOINCREMENT, " +  
 "name TEXT NOT NULL, " +  
 "age INTEGER, " +  
 "sex TINYINT, " +  
 "class TEXT)");  
 }  
  
 @Override  
 public void onUpgrade(RdbStore rdbStore, int i, int i1) {  
 *// 更新数据库时回调* }  
 });  
  
 return store;  
}

// 插入数据

RdbStore store = getTestRdbStore();  
  
ValuesBucket values = new ValuesBucket();  
values.putString("name", "董昱");  
values.putInteger("age", 30);  
values.putInteger("sex", 1);  
values.putString("class", "鸿蒙基础学习班");  
long id = store.insert("student", values);  
if (id == -1) {  
 Utils.*showToast*(this, "数据插入失败!");  
} else {  
 Utils.*showToast*(this, "数据插入成功，Row ID: " + id + "!");  
}  
*// 使用完毕后要关闭RdbStore对象*store.close();

// 新增数据

RdbStore store = getTestRdbStore();  
  
List<ValuesBucket> students = new ArrayList<>();  
ValuesBucket student1 = new ValuesBucket();  
student1.putString("name", "王娜");  
student1.putInteger("age", 30);  
student1.putInteger("sex", 0);  
student1.putString("class", "鸿蒙基础学习班");  
students.add(student1);  
  
ValuesBucket student2 = new ValuesBucket();  
student2.putString("name", "董沐宸松");  
student2.putInteger("age", 20);  
student2.putInteger("sex", 1);  
student2.putString("class", "鸿蒙基础学习班");  
students.add(student2);  
  
List<Long> ids = store.batchInsertOrThrowException("student", students,  
 RdbStore.ConflictResolution.*ON\_CONFLICT\_ABORT*);  
String output = "";  
for(long id : ids) {  
 if (id == -1) {  
 Utils.*log*("数据插入失败!");  
 } else {  
 Utils.*log*("数据插入成功，Row ID: " + id + "!");  
 }  
}  
*// 使用完毕后要关闭RdbStore对象*store.close();

// 查询数据

RdbStore store = getTestRdbStore();  
  
String[] columns = new String[] {"id", "name", "age", "sex", "class"};  
RdbPredicates rdbPredicates = new RdbPredicates("student")  
 .orderByAsc("id");  
  
ResultSet resultSet = store.query(rdbPredicates, columns);  
  
if (resultSet == null) {  
 Utils.*log*("查询失败!");  
}  
  
Utils.*log*("结果集行数： " + resultSet.getRowCount());  
while (resultSet.goToNextRow()) {  
 int id = resultSet.getInt(0);  
 String name = resultSet.getString(1);  
 int age = resultSet.getInt(2);  
 int sex = resultSet.getInt(3);  
 String \_class = resultSet.getString(4);  
 Utils.*log*("id:"+id+" 姓名："+name+" 年龄："+ age  
 + " 性别：" + (sex == 1 ? "男性" : "女性")  
 + " 班级：" + \_class);  
}  
*// 关闭结果集*resultSet.close();  
*// 关闭RdbStore对象*store.close();

// 更新数据

RdbStore store = getTestRdbStore();  
  
RdbPredicates rdbPredicates = new RdbPredicates("student").equalTo("name", "董昱");  
ValuesBucket values = new ValuesBucket();  
values.putString("class", "鸿蒙高级学习班");  
int rowNum = store.update(values, rdbPredicates);  
if (rowNum == -1) {  
 Utils.*showToast*(this, "数据更新失败!");  
} else {  
 Utils.*showToast*(this, "数据更新成功，更新行数: " + rowNum + "!");  
}  
*// 使用完毕后要关闭RdbStore对象*store.close();

// 删除数据

RdbStore store = getTestRdbStore();  
  
RdbPredicates rdbPredicates = new RdbPredicates("student").equalTo("name", "董昱");  
int rowNum = store.delete(rdbPredicates);  
if (rowNum == -1) {  
 Utils.*showToast*(this, "数据删除失败!");  
} else {  
 Utils.*showToast*(this, "数据删除成功，删除行数: " + rowNum + "!");  
}  
  
store.close();

// 删除数据库DatabaseHelper helper = new DatabaseHelper(this);  
 *// 删除数据库* boolean flag = helper.deleteRdbStore("test.sqlite");  
 if (flag) {  
 Utils.*showToast*(this, "数据库删除成功!");  
 } else {  
 Utils.*showToast*(this, "数据库删除失败!");  
 }

1. 结果分析（原始图示，测试数据与运行记录，分析正确性）

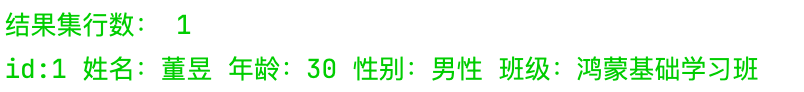


图1 向数据库中插入数据

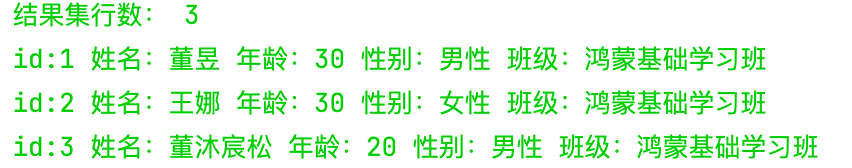


图2 向数据库中批量插入数据



图3 修改数据库中的数据

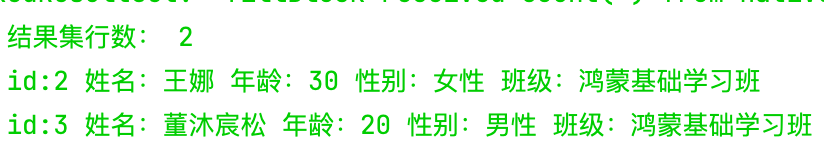


图 4 删除数据



图 5删除数据库

1. 调试报告

将数据存储在本地的数据库中保存一些隐私性较低的数据供应用使用降低后端压力以及提高用户使用流畅度。

1. 实验内容2
2. 问题描述

了解鸿蒙偏好数据库存储的相关基础知识并使用鸿蒙开发中偏好型数据库的相关操作。

1. 设计思想与原理

轻量级数据存储适用于对Key-Value结构的数据进行存取和持久化操作。应用获取某个轻量级存储对象后，该存储对象中的数据将会被缓存在内存中，以便应用获得更快的数据存取速度。应用也可以将缓存的数据再次写回文本文件中进行持久化存储，由于文件读写将产生不可避免的系统资源开销，建议应用减少对持久化文件的读写频率。

1. 程序清单（含必要注释说明）

// 读取指定文件，将数据加载到Preferences实例

Context context = getContext(); // 数据文件存储路径：/data/data/{PackageName}/{AbilityName}/preferences。

// Context context = getApplicationContext(); // 数据文件存储路径：/data/data/{PackageName}/preferences。

DatabaseHelper databaseHelper = new DatabaseHelper(context); // context入参类型为ohos.app.Context。

String fileName = "test\_pref"; // fileName表示文件名，其取值不能为空，也不能包含路径，默认存储目录可以通过context.getPreferencesDir()获取。

Preferences preferences = databaseHelper.getPreferences(fileName);

// 使用Preferences put方法保存数据到缓存的实例中。

preferences.putInt("intKey", 3);

preferences.putString("StringKey", "String value");

// 使用Preferences get方法读取数据。

int value = preferences.getInt("intKey", 0);

// 应用存入数据到Preferences实例后，可以通过flush或者flushSync方法将Preferences实例回写到文件中。

preferences.flush(); // 异步方法

bool result = preferences.flushSync(); // 同步方法

//应用订阅数据变化需要指定PreferencesObserver作为回调方法。订阅的key的值发生变更后，当执行flush方法时，PreferencesObserver被触发回调。不再需要PreferencesObserver时请注销。

private class PreferencesObserverImpl implements Preferences.PreferencesObserver {

@Override

public void onChange(Preferences preferences, String key) {

if ("intKey".equals(key)) {

HiLog.info(LABLE, "Change Received：[key=value]");

}

}

}

// 订阅数据变化

PreferencesObserverImpl observer = new PreferencesObserverImpl();

preferences.registerObserver(observer);

// 修改数据

preferences.putInt("intKey", 3);

// 触发订阅者回调方法执行

preferences.flush();

// 修改数据后，observer的onChange方法会被回调

// 注销订阅

preferences.unRegisterObserver(observer);

//从内存中移除指定文件对应的Preferences单实例。移除Preferences单实例时，应用不允许再使用该实例进行数据操作，否则会出现数据一致性问题。

DatabaseHelper databaseHelper = new DatabaseHelper(context);

String fileName = "name"; // fileName表示文件名，其取值不能为空，也不能包含路径。

databaseHelper.removePreferencesFromCache(fileName);

// 使用deletePreferences方法从内存中移除指定文件对应的Preferences单实例，并删除指定文件及其备份文件、损坏文件。删除指定文件时，应用不允许再使用该实例进行数据操作，否则会出现数据一致性问题。删除后，数据及文件将不可恢复。

DatabaseHelper databaseHelper = new DatabaseHelper(context);

String fileName = "name"; // fileName表示文件名，其取值不能为空，也不能包含路径。

boolean result = databaseHelper.deletePreferences(fileName);

// 从源路径移动文件到目标路径。移动文件时，应用不允许再操作该文件数据，否则会出现数据一致性问题。

Context targetContext = getContext();

DatabaseHelper databaseHelper = new DatabaseHelper(targetContext);

String srcFile = "srcFile"; // srcFile表示源文件名或者源文件的绝对路径，不能为相对路径，其取值不能为空。当srcFile只传入文件名时，srcContext不能为空。

String targetFile = "targetFile"; // targetFile表示目标文件名，其取值不能为空，也不能包含路径。

Context srcContext = getApplicationContext();

boolean result = databaseHelper.movePreferences(srcContext, srcFile, targetFile);

1. 结果分析（原始图示，测试数据与运行记录，分析正确性）

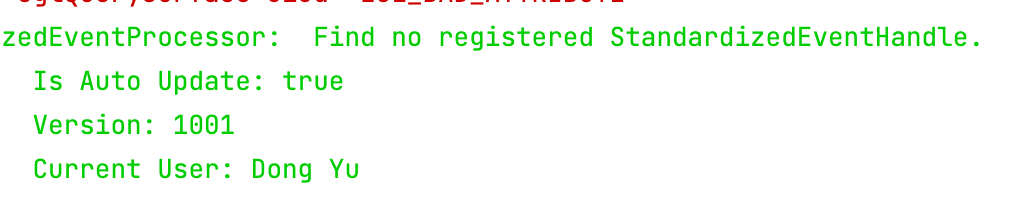


图 1 将数据以键值对的形式存储

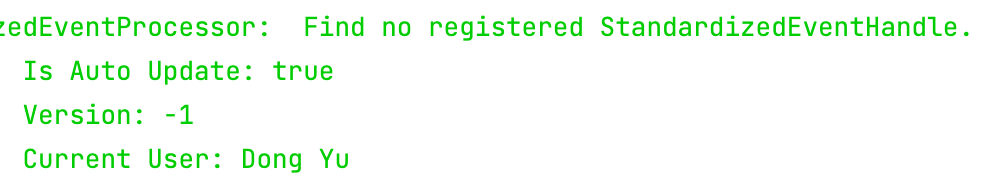


图 2 删除键值对

1. 调试报告

将数据根据key-value持久化地存储在设备上，常用于保存应用的一些常用配置信息

1. 实验内容3
2. 问题描述

练习本地文件存储方法的操作

1. 设计思想与原理

数据存储管理指导开发者基于HarmonyOS进行存储设备（包含本地存储、SD卡、U盘等）的数据存储管理能力的开发，包括获取存储设备列表，获取存储设备视图等。为了给用户展示存储设备信息，开发者可以使用数据存储管理接口获取存储设备视图信息，也可以根据用户提供的文件名获取对应存储设备的视图信息。

1. 程序清单（含必要注释说明）

// 调用查询设备视图接口。

// 获取默认存储设备挂载状态

MountState status = DataUsage.getDiskMountedStatus();

// 获取存储设备列表

Optional<List<Volume>> list = DataUsage.getVolumes();

// 默认存储设备是否为可插拔设备

boolean pluggable = DataUsage.isDiskPluggable();

// 查询设备视图属性

// 调用查询设备视图接口获取某个设备视图Volume。

// 调用Volume的接口即可查询视图属性。

static final HiLogLabel LABEL = new HiLogLabel(HiLog.LOG\_APP, 0x00201, "MY\_TAG");

// 获取example.txt 文件所在的存储设备的视图属性

Optional<Volume> volume = DataUsage.getVolume(new File("/sdcard/example.txt"));

volume.ifPresent(theVolume -> {

HiLog.info(LABEL, "isEmulated: %{public}t", theVolume.isEmulated());

HiLog.info(LABEL, "isPluggable: %{public}t", theVolume.isPluggable());

HiLog.info(LABEL, "Description: %{public}s", theVolume.getDescription());

HiLog.info(LABEL, "Volume UUID: %{public}d", theVolume.getVolUuid());

}

);

1. 结果分析（原始图示，测试数据与运行记录，分析正确性）

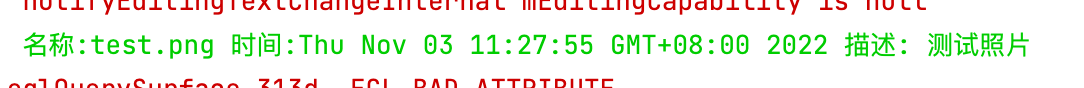


图 1 插入本地数据



图 2 更新数据



图 3 删除数据

1. 调试报告

可以将用户的一些个人静态数据存储在本地文件内，减少用户对后端的访问。

请注意报告排版格式：

页面排版紧凑