

sqflite 数据库，数据库的 CRUD 操作，数据库的事务和批处理

一巴掌拍出两坨脂肪

本章知识点主要介绍 Flutter 的数据持久化之数据库。很多时候我们的数据并不是单一结构且存在关系性，并对大批量数据有增、删、改、查 操作时那么对数据库的使用那就是必不可少。

本章简要：

- 1、sqflite 数据库
- 2、数据库的 CRUD 操作
- 3、数据库的事务和批处理

一、sqflite 数据库

sqflite 数据库是一款轻量级的关系型数据库，如同 iOS 和 Android 中的 SQLite。sqflite 地址：<https://github.com/tekartik/sqflite>

sqflite 插件引入

- 1、在 pubspec.yaml 文件中添加依赖

```
dependencies:  
  fluttertoast: ^3.0.3  
  flutter:  
    sdk: flutter  
  #添加持久化插件 sp  
  shared_preferences: ^0.5.3+1
```

```
#添加文件库
path_provider: ^1.2.0
#添加数据库
sqflite: ^1.1.6
```

本人使用的当前最新版本 1.1.6，读者想体验最新版本请在使用时参看最新版本号进行替换。

2、安装依赖库

执行 `flutter packages get` 命令；AS 开发工具直接右上角 `packages get` 也可。

3、在需要使用地方导包引入

```
import 'package:sqflite/sqflite.dart';
```

sqflite 支持的数据类型

存储类	描述
NULL	值是一个 NULL 值。
INTEGER <small>integer 整数</small>	值是一个带符号的整数， -2^{63} 到 $2^{63} - 1$
REAL <small>real 真的，非常</small>	值是一个数字类型，dart中的 num
TEXT	值是一个文本字符串，dart中的 String
BLOB <small>blob 一滴，一抹</small>	值是一个 blob 数据，dart中的 Uint8List 或者 List<int>

可以看出 sqflite 中支持的数据类型比较少，比如 `bool`、`DateTime` 都是不支持的；开发中需要 `bool` 类型可以使用 `INTEGER` 的 0 和 1 来表示，`DateTime` 类型可以使用 时间戳字符串。

sqlite 中常用的 API:

`getDatabasesPath()` : 获取数据库位置，在Android上，它通常是 `data/data/包名/databases`；在iOS上，它是 `Documents` 目录。

`join("参数1", "参数2")`: 该方法表示创建数据库，参数1: `getDatabasesPath()` 获取到的数据库存放路径，参数2: 数据库的名字，如: `User.db`

`openDatabase()`: 该方法表示打开数据，具体有以下几个重要参数

```
Future<Database> openDatabase(  
    String path,  
    {int version,  
    OnDatabaseConfigureFn onConfigure,  
    OnDatabaseCreateFn onCreate,  
    OnDatabaseVersionChangeFn onUpgrade,  
    OnDatabaseVersionChangeFn onDowngrade,  
    OnDatabaseOpenFn onOpen,  
    bool readOnly = false,  
    bool singleInstance = true})
```

`path`: 必传参数，`join()` 创建数据库后的返回值。

`version`: 当前的版本号。

`onConfigure`: 数据库的相关配置

`onCreate`: 创建表的方法

onUpgrade、onDowngrade: 数据库版本的升降级

readOnly: 是否为只读方式打开。

CRUD 相关 API:

1、插入数据的两种方式:

```
Future<int> insert(String table, Map<String, dynamic>
values,
    {String nullColumnHack, ConflictAlgorithm
conflictAlgorithm});
```

```
Future<int> rawInsert(String sql, [List<dynamic>
arguments]);
```

`insert()` 方法第一个参数为操作的表名, 第二个参数 `map` 中
是想要添加的字段名和对应字段值。

`rawInsert()` 方法第一个参数为一条插入 `sql` 语句, 语句中 `?` 作
为占位符, 通过第二个参数填充占位数据。

2、查询数据的两种方式:

```
Future<List<Map<String, dynamic>>> query(String table,
    {bool distinct,
    List<String> columns,
    String where,
    List<dynamic> whereArgs,
    String groupBy,
    String having,
    String orderBy,
    int limit,
```

```
int offset});  
  
Future<List<Map<String, dynamic>>> rawQuery(String sql,  
    [List<dynamic> arguments]);
```

2.1 query() 方式查询的参数介绍:

参数	描述
table	表名
distinct	是否去重
columns	查询字段集合
where	WHERE子句 (使用?作为占位符)
whereArgs	WHERE子句占位符参数值
groupBy	结果集分组
having	结合groupBy使用过滤结果集
orderBy	排序方式
limit	查询的条数
offset	查询的偏移位

2.2 `rawQuery()` 方法第一个参数为一条查询 sql 语句，使用 ? 作为占位符，通过第二个参数填充数据。

3. 修改数据的两种方式

```
Future<int> update(String table, Map<String, dynamic>  
values,  
    {String where,
```

```

        List<dynamic> whereArgs,
        ConflictAlgorithm conflictAlgorithm});

Future<int> rawUpdate(String sql, [List<dynamic>
arguments]);

```

`update()`方法第一个参数为操作的表名，第二个参数为修改的字段和对应值，后边的可选参数依次表示WHERE子句、WHERE子句占位符参数值、发生冲突时的操作算法（包括回滚、终止、忽略等）。

`rawUpdate()`方法第一个参数为一条更新sql语句，使用?作为占位符，通过第二个参数填充数据。

4. 删除数据的两种方式

```

Future<int> delete(String table, {String where,
List<dynamic> whereArgs});

Future<int> rawDelete(String sql, [List<dynamic>
arguments]);

```

`delete()`方法第一个参数为操作的表名，后边的可选参数依次表示WHERE子句、WHERE子句占位符参数值。

`rawDelete()`方法第一个参数为一条删除sql语句，使用?作为占位符，通过第二个参数填充数据。

`close()` 关闭数据库。

`transaction()` 开启事务。 transaction 交易，业务；办理，处理

`batch()` 获取批处理对象。 batch 一批，分批处理

可以看到 Flutter 在 增、删、改、查 中都提供了两套方法使用，

更倾向于写 sql 语句的客官们使用 `rawxxx` 方式就比较好；但我还是喜欢通过参数拼接组合的方式，可以屏蔽很多细节问题。

二、数据库的 CRUD

上面对 sqlite 引入和相关 API 都做了介绍，如果还是不清楚怎么使用，接下来就通过 案例的方式去学习。为了不重复的叙述，这里需要提前先做两个准备工作：① 新建一个用户实体类（方便数据操作）。② 建库、建表。

用户实体类

```
class UserInfo{
    String name;
    String password;
    int age;
    UserInfo({this.name,this.age,this.password});
    UserInfo.toUser(Map<String, dynamic> json) {
        name = json['name'];
        age = json['age'];
        password = json['password'];
    }

    Map<String, dynamic> toMap() {
        Map<String, dynamic> data = new Map<String,
dynamic>();
        data['name'] = this.name;
        data['age'] = this.age;
        data['password'] = this.password;
        return data;
    }
}
```

```
}  
}
```

建库、建表

```
import 'package:sqflite/sqflite.dart';  
import 'package:path/path.dart';  
import 'UserInfo.dart';  
  
class SqlUserHelper{  
  //数据库  
  final String dataBaseName = "User.db";  
  
  //数据表  
  final String tableName = "USER_TABLE";  
  
  //以下是表中的列名  
  final String columnId = 'id';  
  final String name = 'name';  
  final String password = 'password';  
  final String age ="age";  
  
  // 静态私有成员  
  static SqlUserHelper _instance;  
  
  Database _database;  
  // 私有构造函数  
  SqlUserHelper._() {  
  
    initDb();  
  }  
  
  //私有访问点  
  static SqlUserHelper helperInstance() {  
    if (_instance == null) {
```



```

        _instance = SqlUserHelper.__(__);
    }
    return _instance;
}
//初始化数据库
void initDb() async {
    String databasesPath = await getDatabasesPath();
    String path = join(databasesPath, dbName );
    // openDatabase 指定是数据库路径，版本号，和执行表的创建
    _database = await openDatabase(path, version: 1,
onCreate: _onCreate);
}
//创建UserInfo表
void _onCreate(Database db, int newVersion) async {
    await db.execute('CREATE TABLE $tableName($columnId
INTEGER PRIMARY KEY AUTOINCREMENT, $name TEXT, $password
TEXT, $age INTEGER)');
}
///关闭数据库
Future<void> close() async {
    return _database.close();
}
}

```

UserInfo 实体比较简单只声明了三个属性而已，另外两个方法只是为了插入数据和查询时方便观看而已。SqlUserHelper 是一个单例模式的用户数据库操作类 (单例模式不清楚的请看上一章节)。在使用完数据库时一定要及时关闭数据库，避免造成不必要的资源浪费。不持续叨逼叨.... 代码中已有详细注释。

1、添加数据

首先向数据库插入几条数据，方便后面查询使用。在 SqlUserHelper 中添加插入数据的方法。

插入数据的两种方式

```
/// insert第一种
Future<int> insert(UserInfo userInfo){
    return _database.insert(tableName,
userInfo.toMap());
}

/// insert第二种
Future<int> rawInsert(UserInfo userInfo){
    return _database.rawInsert("INSERT INTO $tableName
($name,$password,$age) VALUES(?, ?, ?)",
[userInfo.name,userInfo.password,userInfo.age]);
}
```

实例代码：

```
//构建一个user 对象
UserInfo user = UserInfo(name: userName,password:
userPass,age: age);

//向数据库插入该条数据
sqlUserHelper.insert(user).then((value){
    print("the last insert id $value");
});

//构建一个user 对象
UserInfo user = UserInfo(name: userName,password:
userPass,age: age);

//向数据库插入该条数据
```

```
sqlUserHelper.rawInsert(user).then((value){  
    print("the last rawInsert id $value");  
});
```

无论是通过 `insert` 还是 `rawInsert` 方式插入数据，只要成功插入就会返回最后一条插入的记录 ID 回来。

实操演示：

请输入姓名

您的密码

您的年龄

添加(insert) 添加(rawInsert)

控制台输出：

```
I/flutter: the last insert id 1  
I/flutter: the last rawInsert id 2
```

控制台打印出了通过两种方式插入数据后的记录 ID，证明此时已经成功插入了两条一样的数据到数据库。

2、查询数据

查询数据的两种方式

```
///第一种 query  
Future<List<Map>> query() async {  
    List<Map> maps = await _database.query(tableName);  
    if (maps.isNotEmpty) {  
        return maps;  
    }  
}
```

```

        return null;
    }

    ///第二种 query
    Future<List<Map>> rawQuery() async {
        List<Map> maps = await _database.rawQuery("SELECT *
FROM $tableName");
        if (maps.isNotEmpty) {
            return maps;
        }
        return null;
    }

```

实例代码：

```

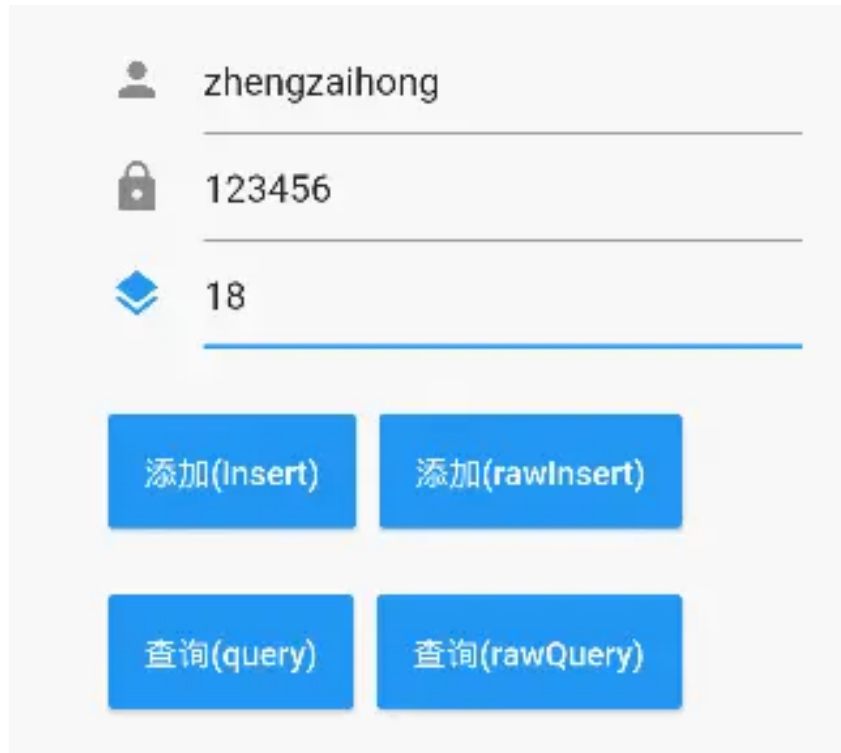
///查询全部
sqlUserHelper.query().then((value){
    print("the query info  ${value.toString()}");
});

///查询全部
sqlUserHelper.rawQuery().then((value){
    print("the rawQuery info  ${value.toString()}");
});

```

两种方式都是查询整个 user 表中的全部信息，并将结果打印输出。下面来查询上面添加的数据。

实操演示：



zhengzaihong

123456

18

添加(Insert) 添加(rawInsert)

查询(query) 查询(rawQuery)

可以看到在我添加数据后，分别点了 `query` 和 `rawQuery` 方式查询，控制台输出结果如下：

```
I/flutter: the query info [{id: 1, name: zhengzaihong, password: 123456, age: 18}, {id: 2, name: zhengzaihong, password: 123456, age: 18}]  
  
I/flutter: the rawQuery info [{id: 1, name: zhengzaihong, password: 123456, age: 18}, {id: 2, name: zhengzaihong, password: 123456, age: 18}]
```

两种方式都查询出了 全部的信息，实际开发中一般都是 **条件查询**，这里只是简单除暴的演示而已。

3、修改数据

上面有了两条一样的数据，接下来方便我们做修改，然后再次查询输出结果看是否正确修改。

修改数据的两种方式

```

    ///第一种 update
    Future<int> update(UserInfo user,int id) async {
        return await
_database.update(tableName,user.toMap(),where: '$columnId
= ?', whereArgs: [id]);
    }

    ///第二种 rawUpdate
    Future<int> rawUpdate(UserInfo user,int id) async {
        return await _database.rawUpdate("UPDATE $tableName
SET $name = ? WHERE $columnId = ? ",[user.name,id]);
    }

```

实例代码：


```


    //构建一个user 对象 根据 ID 修改
    UserInfo user = UserInfo(name: userName,password:
userPass,age: age);
    sqlUserHelper.update(user, 1).then((value){
        print("the update info  ${value.toString()}");
    });


    UserInfo user = UserInfo(name: userName,password:
userPass,age: age);
    sqlUserHelper.rawUpdate(user, 2).then((value){
        print("the rawUpdate info  ${value.toString()}");
    });

```

实操演示

 zhengzaihong

 123456

 18

添加(insert) 添加(rawInsert)

查询(query) 查询(rawQuery)

修改(update) 修改(rawUpdate)

控制台输出：

```
I/flutter: the update info 1
I/flutter: the rawUpdate info 1
I/flutter: the query info [{id: 1, name: zzh, password: 123456, age: 20}, {id: 2, name: LQ, password: 123456, age: 18}]
I/flutter: the rawQuery info [{id: 1, name: zzh, password: 123456, age: 20}, {id: 2, name: LQ, password: 123456, age: 18}]
```

从输出可以看出 无论是 `update` 还是 `rawUpdate` 方式修改数据

都打印出有一行受影响，这表示数据被成功修改了。update 方式把输入框中的数据重新带入把 id =1 的这条数据全部重新赋值一遍，实现了数据修改；而 rawUpdate 方式只在 sql 语句中加入了对名字的修改，可以看到在输入框填写了年龄值也并未修改成功。

4、删除数据

删除数据的两种方式

```
///第一种 delete 根据id删除
Future<int> delete(int id) async {
    return await _database.delete(tableName,
        where: "$columnId = ?", whereArgs: [id]);
}

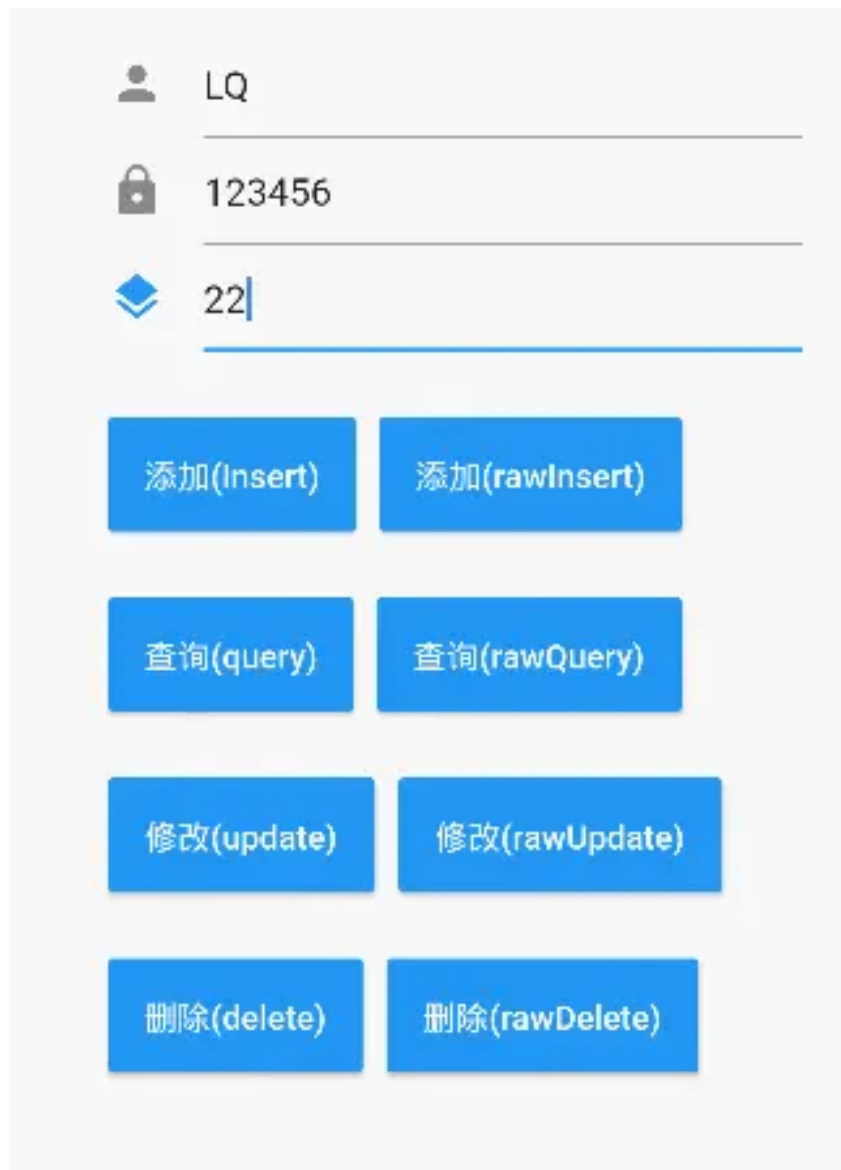
///第二种 delete 根据id删除
Future<int> rawDelete(int id) async {
    return await _database.rawDelete("DELETE FROM
$tableName WHERE $columnId = ?", [id]);
}
```

实例代码：

```
/// 根据 id 删除
sqlUserHelper.delete(1).then((value){
    print("the delete info  ${value.toString()}");
});

/// 根据 id 删除
sqlUserHelper.rawDelete( 2).then((value){
    print("the rawDelete info  ${value.toString()}");
});
```


实操演示：



Flutter application interface showing input fields and operation buttons:

- Username: LQ
- Password: 123456
- Card Number: 22

Buttons available:

- 添加(insert)
- 添加(rawInsert)
- 查询(query)
- 查询(rawQuery)
- 修改(update)
- 修改(rawUpdate)
- 删除(delete)
- 删除(rawDelete)

在我点击两次删除按钮后，再次查询结果如下：

```
I/flutter: the delete info 1
```

```
I/flutter: the rawDelete info 1
```

```
I/flutter: the query info null
```

查询结果输出为 `null` 表示数据库中已经没有开始存入的两条

数据了，证明两次删除都是成功的。

三、数据库的事务和批处理

1、事务

sqlite 同时支持事务，所谓事务，它是一个操作序列，这些操作要么都执行，要么都不执行，即：执行单个逻辑功能的一组指令或操作称为事务。

下面通过事务来添加两条数据： transaction 交易，事务

```
/// 开启事务添加
Future<bool> transactionInsert(UserInfo userInfo1,
UserInfo userInfo2) async {
    return await _database.transaction((Transaction
transaction) async {

        int id1 = await transaction.insert(tableName,
userInfo1.toMap());

        int id2 = await transaction.insert(tableName,
userInfo2.toMap());

        return id1 != null && id2 != null;
    });
}
```

实例代码：

```
//构建两个用户对象

UserInfo user1 = UserInfo(name: "zhengxian",password:
"123456",age: 18);

UserInfo user2 = UserInfo(name: "zzh",password:
```

```
"123456",age: 20);

sqlUserHelper.transactionInsert(user1,
user2).then((value){
    print("transaction result: $value");
});
```

这里就不贴图了，下面直接来看通过事务来添加的两条数据和查询结果，控制台输出：

```
I/flutter: transaction result: true
I/flutter: the query info [{id: 1, name: zhengxian,
password: 123456, age: 18}, {id: 2, name: zzh, password:
123456, age: 20}]
```

结果返回 true ,查询结果也是上面构建的两条数据信息，说明都是成功的。

2、批处理

sqlite 支持批处理操作，批处理指的是一次操作中执行多条 SQL 语句，批处理相比于一次一次执行效率会提高很多。

下面通过批处理来新增一条数据和修改一条数据 batch 一批，分批处理

```
/// 批处理
Future<List<dynamic>> batch(UserInfo user,UserInfo
user2) async {

    Batch batch = _database.batch();
    //先添加一条数据
    batch.insert(tableName, user.toMap());
    //修改 id 为1的值
    batch.update(tableName,user2.toMap(),where: '$columnId
= ?', whereArgs: [1]);
```

```
return batch.commit();  
}
```

实例代码：

```
UserInfo user1 = UserInfo(name: "zhengxian",password:  
"123456",age: 18);  
  
UserInfo user2 = UserInfo(name: "LiShi",password:  
"123456",age: 55);  
  
sqlUserHelper.batch( user1,user2).then((value){  
  print("the batch info  ${value.toString()}");  
});
```

控制台打印批处理和查询结果：

```
I/flutter: the batch info  [3, 1]  
I/flutter: the query info  [{id: 1, name: LiShi,  
password: 123456, age: 55}, {id: 2, name: zzh, password:  
123456, age: 20},  
  {id: 3, name: zhengxian, password: 123456, age:  
18}]
```

[3, 1] 两个数字分别表示最后一次插入数据后的记录id，和有一行修改受影响。通过查询我可以看出，数据确实成功插入到了数据库，且 id=1 的这条数据被成功修改了。

下面贴出下数据库操作的源码，界面源码就不贴了，代码全部详情会在文末给出：

```
import 'package:sqflite/sqflite.dart';  
import 'package:path/path.dart';  
import 'UserInfo.dart';  
  
class SqlUserHelper{
```

```
//数据库
final String dbName = "User.db";

//数据表
final String tableName = "USER_TABLE";

//以下是表中的列名
final String columnId = 'id'; column 队, 纵行
final String name = 'name';
final String password = 'password';
final String age = "age";

// 静态私有成员
static SqlUserHelper _instance;

Database _database;
// 私有构造函数
SqlUserHelper._() {

    initDb();
}

//私有访问点
static SqlUserHelper helperInstance() {
    if (_instance == null) {
        _instance = SqlUserHelper._();
    }
    return _instance;
}

//初始化数据库
void initDb() async {

    String databasesPath = await getDatabasesPath();
```

```

        String path = join(databasesPath, dataBaseName );

        // openDatabase 指定是数据库路径, 版本号, 和执行表的创建
        _database = await openDatabase(path, version: 1,
onCreate: _onCreate);
    }

    //创建UserInfo表
    void _onCreate(Database db, int newVersion) async {

        await db.execute('CREATE TABLE $tableName($columnId
INTEGER PRIMARY KEY AUTOINCREMENT, $name TEXT, $password
TEXT, $age INTEGER)');
    }

    /// 插入数据的两种方式
    /// insert第一种
    Future<int> insert(UserInfo userInfo){
        print("插入数据: ${ userInfo.toMap()}");
        return _database.insert(tableName, userInfo.toMap());
    }

    /// insert第二种
    Future<int> rawInsert(UserInfo userInfo){
        return _database.rawInsert("INSERT INTO $tableName
($name,$password,$age) VALUES(?, ?, ?)",
[userInfo.name,userInfo.password,userInfo.age]);
    }

    ///查询数据的两种方式
    ///第一种 query
    Future<List<Map>> query() async {
        List<Map> maps = await _database.query(tableName);
        if (maps.isNotEmpty) {
            return maps;
        }
    }

```

```

    }
    return null;
}

///第二种 query
Future<List<Map>> rawQuery() async {
    List<Map> maps = await _database.rawQuery("SELECT *
FROM $tableName");
    if (maps.isNotEmpty) {
        return maps;
    }
    return null;
}

///修改数据的两种方式
///第一种 update
Future<int> update(UserInfo user,int id) async {
    return await
_database.update(tableName,user.toMap(),where: '$columnId
= ?', whereArgs: [id]);
}

///第二种 rawUpdate
Future<int> rawUpdate(UserInfo user,int id) async {
    return await _database.rawQuery("UPDATE $tableName
SET $name = ? WHERE $columnId = ? ",[user.name,id]);
}

///删除数据的两种方式
///
///第一种 delete 根据id删除
Future<int> delete(int id) async {
    return await _database.delete(tableName,
        where: "$columnId = ?", whereArgs: [id]);
}

```

```

    ///第二种 delete 根据id删除
    Future<int> rawDelete(int id) async {
        return await _database.rawDelete("DELETE FROM
$tableName WHERE $columnId = ?", [id]);
    }

    /// 开启事务添加
    Future<bool> transactionInsert(UserInfo userInfo1,
    UserInfo userInfo2) async {
        return await _database.transaction((Transaction
transaction) async {

            int id1 = await transaction.insert(tableName,
userInfo1.toMap());

            int id2 = await transaction.insert(tableName,
userInfo2.toMap());

            return id1 != null && id2 != null;
        });
    }

    /// 批处理
    Future<List<dynamic>> batch(UserInfo user,UserInfo
user2) async {

        Batch batch = _database.batch();
        //先添加一条数据
        batch.insert(tableName, user.toMap());
        //修改 id 为1的值
        batch.update(tableName,user2.toMap(),where:
'$columnId = ?', whereArgs: [1]);

        return batch.commit();
    }

```



```
}

///关闭数据库
Future<void> close() async {
    return _database.close();
}
}
```

至此数据库的一些用法也就介绍完了，加上前一篇文章中介绍的 `ShardPreferences` 和 `IO 文件读写方式` 总共介绍了三种方式可以实现数据持久化。`ShardPreferences` 多用于一些简单无关系性的数据存储；`IO 文件` 的读写其实在开发中很少使用来存储数据，往往都是做一些文档文件才会使用。

好了本章节到此结束，又到了说再见的时候了，如果你喜欢请留下你的小红星，创作真心也不容易；你们的支持才是创作的动力，如有错误，请热心的你留言指正，谢谢大家观看，下章再会 $O(n_n)O$

实例源码地址：https://github.com/zhengzaihong/flutter_learn/tree/master/lib/page/database