sqflite

Sqflite 为flutter插件，支持iOS，Android 及 MacOS

- 支持事务及批量处理
- 打开期间自动管理版本
- 提供插入/查询/更新/删除的小助手
- 在iOS与Android平台上，数据库操作在后台线程执行

其它支持的平台：

- 使用 sqflite_common_ffi 支持Linux/Windows/DartVM
- 不支持Web端 查看详细信息

准备开始

在你的Flutter工程内添加依赖：

```
dependencies:
  ...
  sqflite: ^1.3.0
```

查看在线文档 documentation ，帮助你快速使用flutter

使用案例

导入头文件 sqflite.dart

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

打开一个数据库

SQLite数据库是文件系统通过路径识别的一个文件，如果是相对的，那么这个路径与getDatabasesPath()函数所返回的路径是相对应的，这个路径在Android上的默认是数据库目录，在iOS上对应沙盒目录

```
var db = await openDatabase('my_db.db');
```

有一个基本的迁移机制用来处理数据库打开期间的模式更改

很多应用使用一个数据库并且可能从来不需要去关闭它（它会在应用程序退出后关闭），如果你需要释放资源，你可以关闭这个数据库

```
await db.close();
```

- 查看更多有关打开数据库的信息
- 完整案例

原始SQL查询

执行原始SQL查询的演示代码

```
// Get a location using getDatabasesPath
var databasesPath = await getDatabasesPath();
String path = join(databasesPath, 'demo.db');

// Delete the database
await deleteDatabase(path);

// open the database
Database database = await openDatabase(path, version: 1,
    onCreate: (Database db, int version) async {
  // When creating the db, create the table
  await db.execute(
      'CREATE TABLE Test (id INTEGER PRIMARY KEY, name TEXT, value INTEGER,
num REAL)');
});
// Insert some records in a transaction
await database.transaction((txn) async {
  int id1 = await txn.rawInsert(
      'INSERT INTO Test(name, value, num) VALUES("some name", 1234,
456.789)');
  print('inserted1: $id1');
  int id2 = await txn.rawInsert(
      'INSERT INTO Test(name, value, num) VALUES(?, ?, ?)',
      ['another name', 12345678, 3.1416]);
  print('inserted2: $id2');
});

// Update some record
int count = await database.rawUpdate(
    'UPDATE Test SET name = ?, value = ? WHERE name = ?',
    ['updated name', '9876', 'some name']);
print('updated: $count');

// Get the records
List<Map> list = await database.rawQuery('SELECT * FROM Test');
List<Map> expectedList = [
  {'name': 'updated name', 'id': 1, 'value': 9876, 'num': 456.789},
  {'name': 'another name', 'id': 2, 'value': 12345678, 'num': 3.1416}
];
print(list);
print(expectedList);
assert(const DeepCollectionEquality().equals(list, expectedList));

// Count the records
count = Sqflite
```

```
    .firstIntValue(await database.rawQuery('SELECT COUNT(*) FROM Test'));
assert(count == 2);

// Delete a record
count = await database
    .rawDelete('DELETE FROM Test WHERE name = ?', ['another name']);
assert(count == 1);

// Close the database
await database.close();
```

关于SQL的基本信息请看 这里

SQL助手

使用助手的案例

```
final String tableTodo = 'todo';
final String columnId = '_id';
final String columnTitle = 'title';
final String columnDone = 'done';

class Todo {
  int id;
  String title;
  bool done;
  Map<String, dynamic> toMap() {
    var map = <String, dynamic>{
      columnTitle: title,
      columnDone: done == true ? 1 : 0
    };
    if (id != null) {
      map[columnId] = id;
    }
    return map;
  }
  Todo();
  Todo.fromMap(Map<String, dynamic> map) {
    id = map[columnId];
    title = map[columnTitle];
    done = map[columnDone] == 1;
  }
}
class TodoProvider {
  Database db;
  Future open(String path) async {
    db = await openDatabase(path, version: 1,
        onCreate: (Database db, int version) async {
      await db.execute('''
```

```
create table $tableTodo (
  $columnId integer primary key autoincrement,
  $columnTitle text not null,
  $columnDone integer not null)
''');
    });
  }
  Future<Todo> insert(Todo todo) async {
    todo.id = await db.insert(tableTodo, todo.toMap());
    return todo;
  }

  Future<Todo> getTodo(int id) async {
    List<Map> maps = await db.query(tableTodo,
        columns: [columnId, columnDone, columnTitle],
        where: '$columnId = ?',
        whereArgs: [id]);
    if (maps.length > 0) {
      return Todo.fromMap(maps.first);
    }
    return null;
  }
  Future<int> delete(int id) async {
    return await db.delete(tableTodo, where: '$columnId = ?', whereArgs:
[id]);
  }
  Future<int> update(Todo todo) async {
    return await db.update(tableTodo, todo.toMap(),
        where: '$columnId = ?', whereArgs: [todo.id]);
  }
  Future close() async => db.close();
}
```

读取结果

假设读取的是以下结果

```
List<Map<String, dynamic>> records = await db.query('my_table');
```

返回列表中的Map是只读的

```
// 获取第一条记录
Map<String, dynamic> mapRead = records.first;
// 修改它的内存，这会抛出异常
mapRead['my_column'] = 1;
// Crash... `mapRead` is read-only
```

如果你想在内存中修改它，你需要创建一个新的Map

```
// get the first record
Map<String, dynamic> map = Map<String, dynamic>.from(mapRead);
// Update it in memory now
map['my_column'] = 1;
```

事务

不要在数据库事务对象中访问数据库对象

```
await database.transaction((txn) async {
  // Ok
  await txn.execute('CREATE TABLE Test1 (id INTEGER PRIMARY KEY)');

  // DON'T  use the database object in a transaction
  // this will deadlock!
  await database.execute('CREATE TABLE Test2 (id INTEGER PRIMARY KEY)');
});
```

当这个回调没有抛出异常的时候这个事务会被提交，如果有异常抛出，这个事务会被取消掉。因此可以通过抛出一个异常去回滚一个事务

批量处理支持

为了避免在dart与原生之间来回碰撞，你可以使用批量处理

```
batch = db.batch();
batch.insert('Test', {'name': 'item'});
batch.update('Test', {'name': 'new_item'}, where: 'name = ?', whereArgs:
['item']);
batch.delete('Test', where: 'name = ?', whereArgs: ['item']);
results = await batch.commit();
```

获取结果的每一个操作都需要花费一定的代价（用于插入的id以及用于更新和删除的更改数），特别是在android上执行特别的请求。如果你不在意这个结果并且也不担心在大批量处理下的表现，你可以使用

```
await batch.commit(noResult: true);
```

警告，在一个事务内，这些批量处理不会被提交直到这个事务被提交

```
await database.transaction((txn) async {
  var batch = txn.batch();

  // ...

  // commit but the actual commit will happen when the transaction is
committed
  // however the data is available in this transaction
  await batch.commit();

  //  ...
});
```

默认情况下，批处理一旦遇到错误就会停止（这通常会恢复未提交的更改），您可以忽略错误，即使有一个操作失败了，其它的每一个成功的操作都会被提交

```
await batch.commit(continueOnError: true);
```

表名和列名

通常情况下最好避免使用SQLite关键字作为实体名称，如果使用了以下的任何一个名称：

```
"add","all","alter","and","as","autoincrement","between","case","check","colla
te","commit","constraint","create","default","deferrable","delete","distinct",
"drop","else","escape","except","exists","foreign","from","group","having","if
","in","index","insert","intersect","into","is","isnull","join","limit","not",
"notnull","null","on","or","order","primary","references","select","set","tabl
e","then","to","transaction","union","unique","update","using","values","when"
,"where"
```

助手将转义该名称，如：

```
db.query('table')
```

相当于在表名周围手工添加双引号(这里的名称是 table)

```
db.rawQuery('SELECT * FROM "table"');
```

但是在其他语句中（包括orderBy, where, groupBy），请确保使用双引号正确转义名称。例如，参见下面的列名称"group"没有在列参数中转义，而是在where参数中转义。

```
db.query('table', columns: ['group'], where: '"group" = ?', whereArgs:
['my_group']);
```

支持的 SQLite 类型

还没有对数值进行有效性检测，所以请避免使用不支持的类型

https://www.sqlite.org/datatype3.html

DateTime是不支持的SQLite类型。我个人将它们存储为int (millisSinceEpoch)或string (iso8601)

bool是不支持的SQLite类型。使用整数、0和1值。

More information on supported types here.

INTEGER

- Dart 类型: int
- 支持的数值范围: -2^63 ~ 2^63 - 1

REAL

- Dart 类型: num

TEXT

- Dart 类型: String

BLOB

- Dart 类型: Uint8List

当前问题

- 由于事务在线程中的工作方式，不支持并发读写事务。当前所有的调用都是同步的，事务独占线程。我认为支持并发访问的基础是多次打开数据库，但它只在iOS上工作，因为android重用相同的数据库对象。我个人认为一个原生线程可能是一个潜在的未来解决方案，然而在android上当在一个事务内访问另一个线程内的数据库是会被阻塞的。。。
- 目前整数限制范围为-2^63到2^63 - 1（尽管Android支持较大的数）

More

- 引导