flutter sqflite数据库使用

江水东流

flutter项目里需要存储数据如果比较少可以用SharedPreferences,
如果存储数据比较多 尤其需要筛选的话 我们就得用数据库了

sqflite

lite 清淡的，低盐的

`sqflite` 是一个 Flutter 插件，用于在 Flutter 应用中访问和操作 SQLite 数据库。它提供了一系列方法来执行 SQL 查询、更新数据、以及管理数据库的创建和版本。以下是使用 `sqflite` 的基本步骤：

打开数据库

创建一个函数来打开数据库，如果数据库不存在，则创建它。

```
Future<Database> openDatabase() async {
  final databasePath = await getDatabasesPath();
  final path = join(databasePath, 'my_database.db');

  return openDatabase(
    path,
    onCreate: (db, version) {
      return db.execute(
        "CREATE TABLE users(id INTEGER PRIMARY KEY, name
TEXT, age INTEGER)",
      );
    },
    version: 1,
  );
}
```

插入数据

使用 `insert` 方法向数据库中插入数据。

```
Future<void> insertUser(Database db, User user) async {
```

```
  await db.insert(
    'users',
    user.toMap(),
    conflictAlgorithm: ConflictAlgorithm.replace,
  );
}
```

查询数据

使用 `query` 方法从数据库中查询数据。

```
Future<List<User>> users(Database db) async {
  final List<Map<String, dynamic>> maps = await
db.query('users');

  return List.generate(maps.length, (i) {
    return User(
      id: maps[i]['id'],
      name: maps[i]['name'],
      age: maps[i]['age'],
    );
  });
}
```

更新数据

使用 `update` 方法更新数据库中的数据。

```
Future<void> updateUser(Database db, User user) async {
  await db.update(
    'users',
    user.toMap(),
    where: "id = ?",
    whereArgs: [user.id],
  );
}
```

删除数据

使用 `delete` 方法从数据库中删除数据。

```
Future<void> deleteUser(Database db, int id) async {
  await db.delete(
    'users',
    where: "id = ?",
    whereArgs: [id],
  );
}
```

关闭数据库

在适当的时候关闭数据库。

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

注意事项

- **数据模型**：考虑创建一个类来表示数据库中的数据模型，这将使代码更加清晰易懂。
- **错误处理**：在进行数据库操作时，务必处理可能出现的错误。
- **异步操作**：数据库操作通常是异步的，确保使用 `async` 和 `await`。
- **资源管理**：适当时关闭数据库，避免资源泄漏。

通过使用 `sqflite`，你可以在 Flutter 应用中方便地实现本地数据存储和管理。

数据库升级

在使用 `sqflite` 管理数据库时，随着应用的发展，你可能需要更改数据库的结构，例如添加新的表或更改现有表的列。这通常涉及到数据库的升级。在 `sqflite` 中，你可以通过定义不同的数据库版本和升级逻辑来处理这些变化。

步骤

1. **增加数据库版本号**：在打开数据库时，将版本号增加到新的版本。
2. **处理升级逻辑**：在 `openDatabase` 方法中，使用 `onUpgrade` 参数来定义数据库升级时的行为。

示例

假设你想在现有的数据库中添加一个新表 `orders`。

```
final int newVersion = 2; // 假设当前版本是 2

Database db = await openDatabase(
  path,
  version: newVersion,
  onUpgrade: (Database db, int oldVersion, int newVersion)
async {
    if (oldVersion < 2) {
      // 从版本 1 升级到版本 2
      await db.execute("CREATE TABLE orders(id INTEGER
PRIMARY KEY, amount INTEGER)");
    }
  },
  // 也可以添加 onDowngrade 来处理降级逻辑
);
```

注意事项

- **逐步升级**：如果有多个版本，确保从每个旧版本到新版本的升级逻辑都被正确处理。
- **备份数据**：在进行结构性更改之前，考虑备份重要数据。
- **测试升级逻辑**：在发布新版本之前，确保测试数据库升级逻辑，以避免数据丢失或损坏。
- **版本控制**：合理管理数据库版本，确保与应用版本的兼容性。
- **错误处理**：在执行升级脚本时，确保妥善处理可能出现的错误。

通过在 `sqflite` 中管理数据库版本和升级逻辑，你可以确保应用的数据结构随着应用的发展而顺利演进。
我封装了 sqflite,存一个 model 取出来也是一个 model
需要查询的属性需要自己设置一个独立属性 ,其它属性放到一个 jsonString 里面就行

```
首先建立一个model id是主键 默认自增,也可以用项目里数据属性的一个id代替,就不用自增了.
```

```dart
class NoteModel {
  //唯一di 如果需要查询属性比较多,这里都增加上,并在创建table地方增加属性
 //db.execute( 'create table if not exists $tableName
 // (id integer primary key autoincrement, content text)');

  int? id;
  // 可以存 jsonString,使用时候 转成model
  String? content;

  NoteModel({this.id, this.content});

  ///提供fromJson以方便将数据库查询结果, 转成Dart Model
  NoteModel.fromJson(Map<String, dynamic> json) {
    id = json['id'];
    content = json['content'];
  }

  ///提供toJson以方便在持久化数据的时候使用
  Map<String, dynamic> toJson() {
    final Map<String, dynamic> data = <String, dynamic>{};
    data['id'] = id;
    data['content'] = content;
    return data;
  }
}
```

下面是使用方法

```dart
  //数据库名字
  String dbName = 'test';
  //表名
  String tableName = 'tableName1';

  Database? database;
  JDTableTool? tableTool;
  //初始化
  void _doInit() async {
    // 每次TestDbContentModel结构变化 version +1
    Database? db = storage.dateBaseMap[dbName];
```

```
    if (db != null) {
      database = db!;
    } else {
      database =
          await openDatabase(dbName, version: 1, onCreate:
(db, version) {
        // 在数据库首次创建时执行的操作
      }, onUpgrade: (db, oldVersion, newVersion) async {
        // 在数据库升级时执行的操作
        if (oldVersion == 1 && newVersion == 2) {
          // 如果当前数据库版本为1，目标版本为2，执行操作
        }
      });
    }

    jdLog('database?.getVersion: ${await
database?.getVersion()}');
    if (database != null) {
  // name TEXT, value INTEGER, num REAL; 需要查询的属性 必须单
独写成属性,如果你的map里面有唯一id 就不必要自增了
      tableTool = JDStorageTool(database!, tableName,
          'create table if not exists $tableName (id
integer primary key autoincrement, content text)');
      _loadAll();
      _loadAll();
    }
  }


  void destroy() {
    tableTool?.destroy();
  }

  ///增加数据
  void _doSave(String nameValue, int age) {
    TestDbContentModel contentModel =
        TestDbContentModel.fromJson({'name': nameValue,
'age': age});
    tableTool?.saveNote(NoteModel(content:
jsonEncode(contentModel.toJson())));
    _loadAll();
```

```
  }

  ///查询数据
  void _loadAll() async {
    var list = await tableTool?.getAllNote() ?? [];
    jdLog('list.length--  ${list.length}');
    setState(() {
      noteList = list;
    });
    _getCount();
  }

  ///更新数据
  void _updateContent() {
    if (id == null || name == null) return;
    TestDbContentModel contentModel =
        TestDbContentModel.fromJson({'name': name, 'age':
age});
    var model = NoteModel(id: id, content:
jsonEncode(contentModel.toJson()));
    tableTool?.update(model);
    _loadAll();
  }

  ///删除数据
  void _doDelete(NoteModel model) {
    tableTool?.deleteNote(model.id!);
    _loadAll();
  }

  ///查询列数
  void _getCount() async {
    var count = await tableTool?.getNoteCount() ?? 0;
    setState(() {
      this.count = count;
    });
  }
```

具体的封装实现在 https://gitee.com/kuaipai/my_app.git