10 лекция.

Тема: Flutter: Работа с базами данных в Android Studio, работа с SQLite, MySQL.

План:

1. **Конфигурация проекта**
2. **Создание простой модели**
3. **Класс базы данных**
4. **Основной файл приложения**

Вместо случайного смешивания логики базы данных в приложении основные методы обработки базы данных помещены в lib/services/db.dart для удобства и простоты обслуживания:

```
db.dart
import 'dart:async';
import 'package:flutter_sqlite_demo/models/model.dart';
import 'package:sqflite/sqflite.dart';

abstract class DB {

  static Database _db;

  static int get _version => 1;

  static Future<void> init() async {

    if (_db != null) { return; }

    try {
      String _path = await getDatabasesPath() + 'example';
```

```
            _db = await openDatabase(_path, version: _version, onCreate: onCreate);
        }
        catch(ex) {
            print(ex);
        }
    }

    static void onCreate(Database db, int version) async =>
        await db.execute('CREATE TABLE todo_items (id INTEGER PRIMARY KEY NOT NULL, task STRING, complete BOOLEAN)');

    static Future<list<map<string, dynamic="">>> query(String table) async => _db.query(table);

    static Future<int> insert(String table, Model model) async =>
        await _db.insert(table, model.toMap());

    static Future<int> update(String table, Model model) async =>
        await _db.update(table, model.toMap(), where: 'id = ?', whereArgs: [model.id]);

    static Future<int> delete(String table, Model model) async =>
        await _db.delete(table, where: 'id = ?', whereArgs: [model.id]);
}
</int></int></int></list<map<string,></void>
```

Этот abstract класс, поскольку он не предназначен для создания экземпляра, и требуется только одна его копия в памяти. Внутренне он содержит ссылку на базу данных SQLite в свойстве _db. Номер версии базы данных жестко запрограммирован (1), но в более сложных приложениях версию базы данных

можно использовать для переноса схем базы данных вверх или вниз по версии, чтобы обеспечить развертывание новых функций без необходимости стирать базу данных и начинать с нуля.

Экземпляр базы данных SQLite создается в методе init с использованием имени базы данных для этого проекта example. Если база данных example еще не существует, автоматически вызывается onCreate. Здесь размещаются запросы на создание структуры таблицы. В этом случае у нас есть таблица todo_items с первичным ключом id, а также поля, соответствующие свойствам в приведенном выше классе TodoItem.

Метод query наряду с insert, update и delete определены для выполнения стандартных операций CRUD в базе данных. Они предоставляют простые абстракции и позволяют содержать логику базы данных в этом классе, что может быть чрезвычайно полезно при рефакторинге или выполнении другого обслуживания приложения вместо того, чтобы, например, выполнять поиск и замену строк в нескольких файлах или исправлять странные ошибки, которые появляются при создании простые изменения

На этой лекции мы рассмотрим, как использовать SQLite во Flutter с пакетом sqflite для локального хранения данных приложения. SQLite существует с 2000 года и является популярным выбором для встраивания баз данных в локальные приложения. Для примера проекта мы создадим очень простое приложение «TODO», которое может создавать, обновлять и удалять элементы TODO из базового интерфейса.

Если у вас еще нет Flutter, вы можете получить копию со страницы установки:

Install - Flutter

Select the operating system on which you are installing Flutter:{{site.alert.note}} **Are you on Chrome OS?** If so, see the official [Chrome OS Flutter installation docs!](/docs/get-started/install/chromeos){{site.alert.end}}

https://flutter.dev/docs/get-started/install

Исходный код, используемый в этой статье, доступен на GitHub:

GitHub - kenreilly/flutter-sqlite-demo: Example project demonstrating how to use Flutter with SQLite

Example project demonstrating how to use Flutter with SQLite - kenreilly/flutter-sqlite-demo

1. Конфигурация проекта

Чтобы использовать SQLite в приложении Flutter, первым шагом является включение пакета sqflite в pubspec.yaml проекта, например, так:

```
pubspec.yaml
name: flutter_sqlite_demo
description: Example project demonstrating how to use Flutter with SQLite
version: 1.0.0+1

environment:
  sdk: ">=2.1.0 <3.0.0"

dependencies:
  flutter:
    sdk: flutter
  sqflite: ^1.2.0
  path_provider: ^1.6.0
  cupertino_icons: ^0.1.2

dev_dependencies:
  flutter_test:
    sdk: flutter
```

```
flutter:
  uses-material-design: true
```

Здесь мы указали sqflite версию 1.2.0 или выше и path_provider на 1.6.0 или выше. Помимо этого, проект остается простым, чтобы с ним было легко работать и понимать.

2. Создание простой модели

Для хранения данных простой класс модели данных предоставит необходимые методы для преобразования между дружественным к SQLite форматом данных и объектом, который можно использовать в приложении. Во-первых, абстрактный класс Model будет служить базовым классом для моделей данных. Этот файл находится в lib/models/model.dart:

```
model.dart
abstract class Model {

  int id;

  static fromMap() {}
  toMap() {}
}
```

Класс Model очень прост и создан для удобства, чтобы определить свойства / методы, которые можно ожидать от моделей данных, например id, как показано выше. Это позволяет создавать одну или несколько конкретных моделей данных, которые будут соответствовать этому базовому шаблону проектирования. Для этого приложения класс модели элементов TODO создается в lib/models/todo-item.dart:

```
todo-item.dart
```

```dart
import 'package:flutter_sqlite_demo/models/model.dart';

class TodoItem extends Model {

  static String table = 'todo_items';

  int id;
  String task;
  bool complete;

  TodoItem({ this.id, this.task, this.complete });

  Map<string, dynamic=""> toMap() {

    Map<string, dynamic=""> map = {
      'task': task,
      'complete': complete
    };

    if (id != null) { map['id'] = id; }
    return map;
  }

  static TodoItem fromMap(Map<string, dynamic=""> map) {

    return TodoItem(
      id: map['id'],
      task: map['task'],
      complete: map['complete'] == 1
    );
```

```
  }
}
</string,></string,></string,>
```

Класс TodoItem содержит свойства task и complete а также имеет простой конструктор для создания нового элемента TODO. Для преобразования между экземплярами объектов TodoItem и Map, используемых базой данных, были определены методы toMap и fromMap.

3. Класс базы данных

Вместо случайного смешивания логики базы данных в приложении основные методы обработки базы данных помещены в lib/services/db.dart для удобства и простоты обслуживания:

```
db.dart
import 'dart:async';
import 'package:flutter_sqlite_demo/models/model.dart';
import 'package:sqflite/sqflite.dart';

abstract class DB {

  static Database _db;

  static int get _version => 1;

  static Future<void> init() async {

    if (_db != null) { return; }

    try {
      String _path = await getDatabasesPath() + 'example';
```

```
            _db = await openDatabase(_path, version: _version, onCreate: onCreate);
        }
        catch(ex) {
            print(ex);
        }
    }

    static void onCreate(Database db, int version) async =>
        await db.execute('CREATE TABLE todo_items (id INTEGER PRIMARY KEY NOT NULL, task STRING, complete BOOLEAN)');

    static Future<list<map<string, dynamic="">>> query(String table) async => _db.query(table);

    static Future<int> insert(String table, Model model) async =>
        await _db.insert(table, model.toMap());

    static Future<int> update(String table, Model model) async =>
        await _db.update(table, model.toMap(), where: 'id = ?', whereArgs: [model.id]);

    static Future<int> delete(String table, Model model) async =>
        await _db.delete(table, where: 'id = ?', whereArgs: [model.id]);
}
</int></int></int></list<map<string,></void>
```

Этот abstract класс, поскольку он не предназначен для создания экземпляра, и требуется только одна его копия в памяти. Внутренне он содержит ссылку на базу данных SQLite в свойстве _db. Номер версии базы данных жестко запрограммирован (1), но в более сложных приложениях

версию базы данных можно использовать для переноса схем базы данных вверх или вниз по версии, чтобы обеспечить развертывание новых функций без необходимости стирать базу данных и начинать с нуля.

Экземпляр базы данных SQLite создается в методе init с использованием имени базы данных для этого проекта example. Если база данных example еще не существует, автоматически вызывается onCreate. Здесь размещаются запросы на создание структуры таблицы. В этом случае у нас есть таблица todo_items с первичным ключом id, а также поля, соответствующие свойствам в приведенном выше классе TodoItem.

Метод query наряду с insert, update и delete определены для выполнения стандартных операций CRUD в базе данных. Они предоставляют простые абстракции и позволяют содержать логику базы данных в этом классе, что может быть чрезвычайно полезно при рефакторинге или выполнении другого обслуживания приложения вместо того, чтобы, например, выполнять поиск и замену строк в нескольких файлах или исправлять странные ошибки, которые появляются при создании простые изменения.

4. Основной файл приложения

И последнее, но не менее важное: у нас есть основная логика приложения и UX в lib/main.dart:

```
lib/main.dart
import 'package:flutter/material.dart';
import 'package:flutter_sqlite_demo/models/todo-item.dart';
import 'package:flutter_sqlite_demo/services/db.dart';

void main() async {

  WidgetsFlutterBinding.ensureInitialized();
```

```
  await DB.init();
  runApp(MyApp());
}

class MyApp extends StatelessWidget {

  @override
  Widget build(BuildContext context) {

    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData( primarySwatch: Colors.indigo ),
      home: MyHomePage(title: 'Flutter SQLite Demo App'),
    );
  }
}

class MyHomePage extends StatefulWidget {

  MyHomePage({Key key, this.title}) : super(key: key);

  final String title;

  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<myhomepage> {
```

```
  String _task;

  List<todoitem> _tasks = [];

  TextStyle _style = TextStyle(color: Colors.white, fontSize: 24);

  List<widget> get _items => _tasks.map((item) => format(item)).toList();

  Widget format(TodoItem item) {

    return Dismissible(
      key: Key(item.id.toString()),
      child: Padding(
        padding: EdgeInsets.fromLTRB(12, 6, 12, 4),
        child: FlatButton(
          child: Row(
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: <widget>[
              Text(item.task, style: _style),
              Icon(item.complete == true ? Icons.radio_button_checked : Icons.radio_button_unchecked, color: Colors.white)
            ]
          ),
          onPressed: () => _toggle(item),
        )
      ),
      onDismissed: (DismissDirection direction) => _delete(item),
    );
  }
```

```
void _toggle(TodoItem item) async {

    item.complete = !item.complete;
    dynamic result = await DB.update(TodoItem.table, item);
    print(result);
    refresh();
}

void _delete(TodoItem item) async {

    DB.delete(TodoItem.table, item);
    refresh();
}

void _save() async {

    Navigator.of(context).pop();
    TodoItem item = TodoItem(
        task: _task,
        complete: false
    );

    await DB.insert(TodoItem.table, item);
    setState(() => _task = '' );
    refresh();
}

void _create(BuildContext context) {

    showDialog(
```

```
          context: context,
          builder: (BuildContext context) {
            return AlertDialog(
              title: Text("Create New Task"),
              actions: <widget>[
                FlatButton(
                  child: Text('Cancel'),
                  onPressed: () => Navigator.of(context).pop()
                ),
                FlatButton(
                  child: Text('Save'),
                  onPressed: () => _save()
                )
              ],
              content: TextField(
                autofocus: true,
                decoration: InputDecoration(labelText: 'Task Name', hintText: 'e.g. pick up bread'),
                onChanged: (value) { _task = value; },
              ),
            );
          }
        );
    }

    @override
    void initState() {

        refresh();
        super.initState();
```

```
  }

  void refresh() async {

    List<map<string, dynamic="">> _results = await DB.query(TodoItem.table);
    _tasks = _results.map((item) => TodoItem.fromMap(item)).toList();
    setState(() { });
  }

  @override
  Widget build(BuildContext context) {

    return Scaffold(
      backgroundColor: Colors.black,
      appBar: AppBar( title: Text(widget.title) ),
      body: Center(
        child: ListView( children: _items )
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () { _create(context); },
        tooltip: 'New TODO',
        child: Icon(Icons.library_add),
      )
    );
  }
}
</map<string,></widget></widget></widget></todoitem></myhomepage>
```

Этот файл является стандартным для любого приложения Flutter и определяет базовый внешний вид приложения и его взаимодействия. Во время

инициализации строка WidgetsFlutterBinding.ensureInitialized() обеспечит правильную инициализацию приложения Flutter при инициализации базы данных с помощью await DB.init().

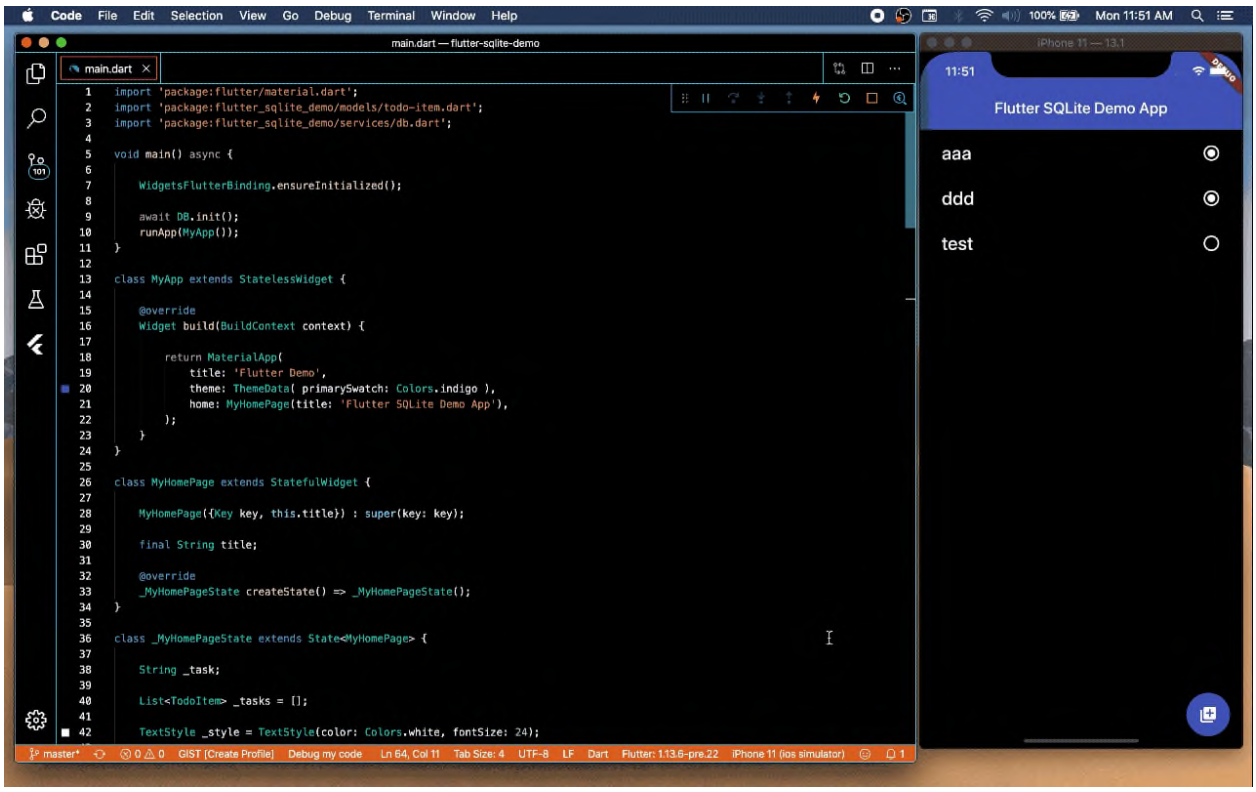
Когда приложение запускается и виджет MyHomePage отображается, делается вызов списка задач с помощью refresh() из таблицы todo_items и сопоставление его с одним из объектов TodoItem. Они будут предоставляться в рамках основного ListView в приложении через аксессор _items, который принимает List из объектов TodoItem и форматирует его в списке виджетов, содержащего текст TODO элемента и индикатор, показывающий, была ли завершена задача.

Задачи могут быть добавлены путем нажатия плавающей кнопки с действием и ввода имени задачи. Когда нажата Save, создается элемент списка с помощью DB.insert. Задача добавляется в базу данных с помощью щелчка по задаче в списке, который переключает логическое значение complete и сохраняя измененный объект обратно в базу данных с DB.update. Свайп по горизонтали на задаче удалит элемент TODO, предоставив его методу DB.delete. Всякий раз, когда в список вносятся изменения, обращение к refresh() и последующему setState() гарантирует, что список обновляется должным образом.

Вывод

SQLite предоставляет удобный стандартный для отрасли способ сохранения данных локально в приложении. В этом примере показано, как реализовать базовые операции CRUD для создания и управления простыми записями в базе данных SQLite. Для получения дополнительной информации о плагине sqflite и различных его функциях см. этот ресурс.

Спасибо за чтение и удачи в вашем следующем проекте Flutter!


Code File Edit Selection View Go Debug Terminal Window Help
100% Mon 11:51 AM
main.dart — flutter-sqlite-demo
main.dart
import 'package:flutter/material.dart';
import 'package:flutter_sqlite_demo/models/todo-item.dart';
import 'package:flutter_sqlite_demo/services/db.dart';
void main() async {
WidgetsFlutterBinding.ensureInitialized();
await DB.init();
runApp(MyApp());
}
class MyApp extends StatelessWidget {
@override
Widget build(BuildContext context) {
return MaterialApp(
title: 'Flutter Demo',
theme: ThemeData(primarySwatch: Colors.indigo),
home: MyHomePage(title: 'Flutter SQLite Demo App'),
);
}
}
class MyHomePage extends StatefulWidget {
MyHomePage({Key key, this.title}) : super(key: key);
final String title;
@override
_MyHomePageState createState() => _MyHomePageState();
}
class _MyHomePageState extends State<MyHomePage> {
String _task;
List<TodoItem> _tasks = [];
TextStyle _style = TextStyle(color: Colors.white, fontSize: 24);
master* 0 0 GIST [Create Profile] Debug my code Ln 64, Col 11 Tab Size: 4 UTF-8 LF Dart Flutter: 1.13.6-pre.22 iPhone 11 (ios simulator) 1
iPhone 11 — 13.1
11:51
DEBUG
Flutter SQLite Demo App
aaa
ddd
test