LeanCode

Persistence in Flutter

Data that persist throughout app lifecycle and beyond

Terminology

SQL database

NoSQL database

- Key-value store
- Object database

Key-value store

```
V get<K, V>(K key);
void put<K, V>(K key, V value);
void delete<K>(K key);
```

In-memory

Key-value store

```
void main() {
  final storage = Storage();

  storage.put('name', 'Jack');
  storage.put('age', 24);

  print(storage['age']);
  storage['name'] = 'Paul';
}

class Storage {
  Storage();

  final _dict = <String, dynamic>{};

  T get<T>(String key) => _dict[key];

  void put<T>(String key, T value) => _dict[key] = value;

  operator [](String key) => _dict[key];

  operator []=(String key, dynamic value) => _dict[key] = value;

}
```

shared_preferences (SharedPreferences / NSUserDefaults)

```dart
import 'package:flutter/material.dart';
import 'package:shared_preferences/shared_preferences.dart';

void main() {
  runApp(MaterialApp(
    home: Scaffold(
      body: Center(
      child: RaisedButton(
        onPressed: _incrementCounter,
        child: Text('Increment Counter'),
        ),
      ),
    ),
  ));
}

_incrementCounter() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  int counter = (prefs.getInt('counter') ?? 0) + 1;
  print('Pressed $counter times.');
  await prefs.setInt('counter', counter);
}
```

flutter_secure_storage (Keystore / Keychain)

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

// Create storage
final storage = new FlutterSecureStorage();

// Read value
String value = await storage.read(key: key);

// Read all values
Map<String, String> allValues = await storage.readAll();

// Delete value
await storage.delete(key: key);

// Delete all
await storage.deleteAll();

// Write value
await storage.write(key: key, value: value);
```

File (IO)

```dart
import 'dart:io';
import 'dart:convert';
import 'dart:async';

void main() async {
  final file = File('file.txt');
  Stream<String> lines = file.openRead()
    .transform(utf8.decoder)       // Decode bytes to UTF-8.
    .transform(LineSplitter());    // Convert stream to individual lines.
  try {
    await for (var line in lines) {
      print('$line: ${line.length} characters');
    }
    print('File is now closed.');
  } catch (e) {
    print('Error: $e');
  }
}
```

SQL based

sqflite

```dart
// 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');
});
```

~~moor~~ drift

```dart
import 'package:drift/drift.dart';
import 'package:drift/native.dart';

part 'database.g.dart';

@DriftDatabase(
  include: {'tables.drift'},
)
class MyDb extends _$MyDb {
  // This example creates a simple in-memory database (without actual
  // persistence).
  // To store data, see the database setups from other "Getting started" guides.
  MyDb() : super(NativeDatabase.memory());

  @override
  int get schemaVersion => 1;
}
```

```
CREATE TABLE todos (
    id INT NOT NULL PRIMARY KEY AUTOINCREMENT,
    title TEXT NOT NULL,
    content TEXT NOT NULL,
    category INTEGER REFERENCES categories(id)
);

CREATE TABLE categories (
    id INT NOT NULL PRIMARY KEY AUTOINCREMENT,
    description TEXT NOT NULL
) AS Category; -- the AS xyz after the table defines the data class name

-- You can also create an index or triggers with drift files
CREATE INDEX categories_description ON categories(description);

-- we can put named sql queries in here as well:
createEntry: INSERT INTO todos (title, content) VALUES (:title, :content);
deleteById: DELETE FROM todos WHERE id = :id;
allTodos: SELECT * FROM todos;
```

```
// inside the database class, the `todos` getter has been created by drift.
@DriftDatabase(tables: [Todos, Categories])
class MyDatabase extends _$MyDatabase {

  // the schemaVersion getter and the constructor from the previous page
  // have been omitted.

  // loads all todo entries
  Future<List<Todo>> get allTodoEntries => select(todos).get();

  // watches all todo entries in a given category. The stream will automatically
  // emit new items whenever the underlying data changes.
  Stream<List<Todo>> watchEntriesInCategory(Category c) {
    return (select(todos)..where((t) => t.category.equals(c.id))).watch();
  }
}
```

NoSQL based

Sembast

(Simple Embedded Application Store database)

```
// File path to a file in the current directory
String dbPath = 'sample.db';
DatabaseFactory dbFactory = databaseFactoryIo;

// We use the database factory to open the database
Database db = await dbFactory.openDatabase(dbPath);
```

```
// dynamically typed store
var store = StoreRef.main();
// Easy to put/get simple values or map
// A key can be of type int or String and the value can be anything as long as it can
// be properly JSON encoded/decoded
await store.record('title').put(db, 'Simple application');
await store.record('version').put(db, 10);
await store.record('settings').put(db, {'offline': true});

// read values
var title = await store.record('title').get(db) as String;
var version = await store.record('version').get(db) as int;
var settings = await store.record('settings').get(db) as Map;

// ...and delete
await store.record('version').delete(db);
```

hive

```
var box = Hive.box('myBox');
box.put('name', 'David');
var name = box.get('name');
print('Name: $name');
```

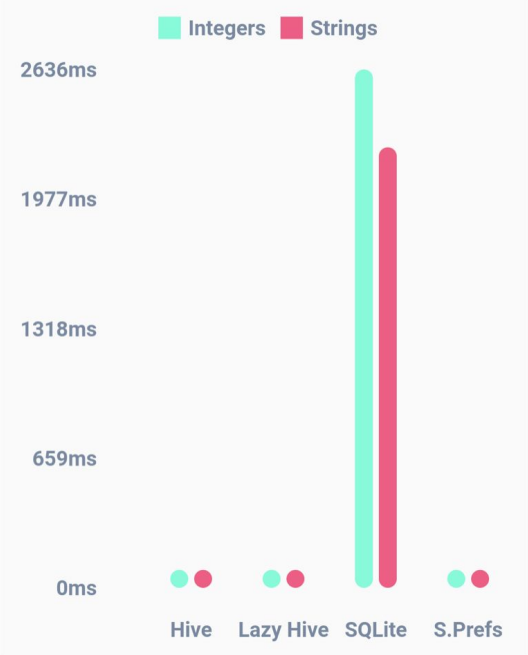
```dart
@HiveType(typeId: 0)
class Person extends HiveObject {

  @HiveField(0)
  String name;

  @HiveField(1)
  int age;
}
```

```dart
var box = await Hive.openBox('myBox');

var person = Person()
  ..name = 'Dave'
  ..age = 22;
box.add(person);

print(box.getAt(0)); // Dave - 22

person.age = 30;
person.save();

print(box.getAt(0)) // Dave - 30
```

```dart
import 'package:hive/hive.dart';
import 'package:hive_flutter/hive_flutter.dart';

class SettingsPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return ValueListenableBuilder(
      valueListenable: Hive.box('settings').listenable(),
      builder: (context, box, widget) {
        return Switch(
          value: box.get('darkMode'),
          onChanged: (val) {
            box.put('darkMode', val);
          }
        );
      },
    );
  }
}
```

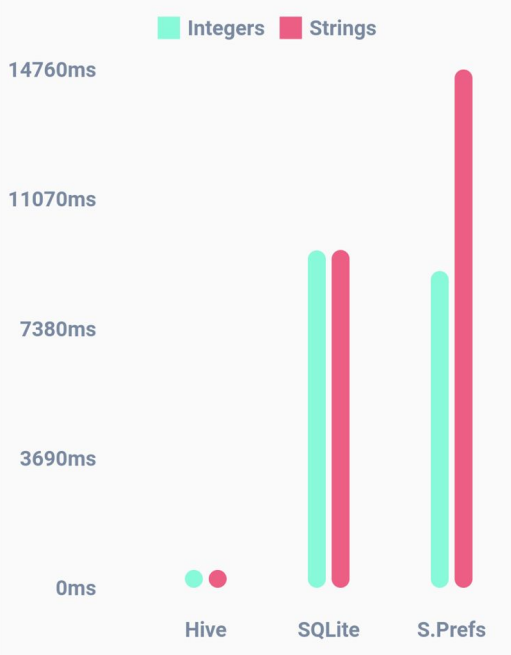
Benchmark

1000 read iterations



SharedPreferences is on par with Hive when it comes to read performance. SQLite performs much worse.

1000 write iterations



Hive greatly outperforms SQLite and SharedPreferences when it comes to writing or deleting.

We build digital products.

LeanCode

More abstraction!

state_persistence

```
return PersistedAppState(
  storage: JsonFileStorage(),
  child: MaterialApp(
    title: 'Persistent TextField Example',
    theme: ThemeData(primarySwatch: Colors.indigo),
    home: Scaffold(
      appBar: AppBar(title: Text('Persistent TextField Example')),
      body: Container(
        padding: const EdgeInsets.all(32.0),
        alignment: Alignment.center,
        child: PersistedStateBuilder(
          builder: (BuildContext context, AsyncSnapshot<PersistedData> snapshot) {
            if (snapshot.hasData) {
              if (_textController == null) {
                _textController = TextEditingController(text: snapshot.data['text'] ?? '');
              }
              return TextField(
                controller: _textController,
                decoration: InputDecoration(
                  hintText: 'Enter some text',
                ),
                onChanged: (String value) => snapshot.data['text'] = value,
              );
```

hydrated_bloc

```
Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  HydratedBloc.storage = await HydratedStorage.build(storageDirectory: ...);
  runApp(App());
}
```

```dart
class CounterCubit extends HydratedCubit<int> {
  CounterCubit() : super(0);

  void increment() => emit(state + 1);

  @override
  int fromJson(Map<String, dynamic> json) => json['value'] as int;

  @override
  Map<String, int> toJson(int state) => { 'value': state };
}
```

Redux_persist, dart_json_mapper_mobx, and so on...

Sources:
docs