сгенерированный код. Его можно изменять, но обычно это не рекомендуется делать, так как в случае какого-либо изменения в файле, по которому генерировался этот код, при повторном запуске команды `dart pub run build_runner build` все внесенные изменения исчезнут. А теперь представьте, что у вас таких мест много и во все вы внесли изменения, модифицировав код. Чем такой подход будет лучше написания сериализации и десериализации вручную? Ничем!!! Можно попросту забыть об этих изменениях, а потом, при очередной пересборке проекта, удивляться, почему он так сейчас работает или не работает вовсе.

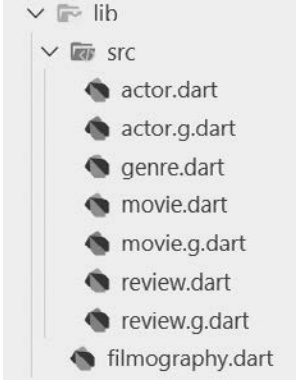

Рис. 5.15. Текущее состояние каталога lib

Чтобы проверить корректность работы приложения, удалите из каталога bin файл `output.json` и запустите программу.

Такой подход позволяет достаточно удобно и быстро реализовывать классы, которые предназначены для получения или отправки данных по сети, хранению в файле и т. д. Да, у него есть ряд ограничений, но куда проще мириться с ними, чем в сотый раз описывать для модели (класса) очередной фабричный конструктор `fromJson` или метод `toJson`.

Если вы хотите более глубоко погрузиться в возможности рассматриваемых пакетов, то обратитесь к их официальной документации, а также можете почитать следующую статью из документации по Flutter: https://docs.flutter.dev/data-and-backend/serialization/json.

5.7. Простая БД по типу «ключ:значение» в формате JSON

В данном разделе главы мы реализуем хранилище по типу «ключ:значение». По сути, мы напишем аналог (мини-версию) достаточно известного Flutter-пакета — `shared_preferences` (https://pub.dev/packages/shared_preferences).

Создайте новый консольный проект `json_store` со следующей структурой каталогов и их наполнением:

```
json_store
├── bin/
│   ├── film.json
│   └── json_store.dart
├── lib/
│   └── src/
│       ├── json_file.dart
│       └── json_store.dart
│   └── json_store.dart
├── test/
└── pubspec.yaml
```

Первым делом откройте `pubspec.yaml` и установите в качестве зависимости пакеты `path` (https://pub.dev/packages/path), что избавит от проблем указания путей

в различных операционных системах, и `collection` (https://pub.dev/packages/collection), функционал которого будем использовать для сравнения двух объектов:

```
dependencies:
  path: ^1.8.3
  collection: ^1.18.0
```

Затем откройте файл `json_file.dart` и добавьте в него следующий код, отвечающий за создание пути каталогов и файла, а также за работу с ним:

```
// json_file.dart
import 'dart:convert';
import 'dart:io';

import 'package:path/path.dart' as p;

class JSONFile {
  final String path;

  JSONFile(this.path) {
    final dir = Directory(p.dirname(path));
    if (!dir.existsSync()) {
      dir.createSync(recursive: true);
    }
  }

  Map<String, Object>? read() {
    final file = File(path);
    try {
      if (file.existsSync()) {
        final data = file.readAsStringSync();
        if (data.isNotEmpty) {
          final json = jsonDecode(data);
          if (json is Map) {
            return json.cast<String, Object>();
          }
        }
      }
    } on FormatException {
      return null;
    }
    return {};
  }

  void write(Map<String, Object> json) {
    final file = File(path);
    if (!file.existsSync()) {
      file.createSync(recursive: true);
    }
    final encoder = const JsonEncoder.withIndent('  ');
    return file.writeAsStringSync(
      encoder.convert(json),
    );
  }
}
```

Так как реализуется простая база данных, которая не подразумевает хранение сериализованных экземпляров классов, то вместо `Map<String, dynamic>` используется `Map<String, Object>`. Тем самым уменьшается набор ситуаций, когда вы можете сами себе выстрелить в ногу и узнать об этом только в процессе работы приложения.

Далее откройте файл `json_store.dart` каталога src. В нем будет объявлен класс `JSONStore`, представляющий собой высокоуровневую обертку над `JSONFile`, с методами для добавления, проверки, замены и извлечения данных из хранилища:

```
// src - json_store.dart
import 'package:collection/collection.dart';

import 'json_file.dart';

class JSONStore {
  final JSONFile _file;
  JSONStore(String path) : _file = JSONFile(path);

  Map<String, Object>? _values;
  String get path => _file.path;

  bool contains(String key) {
    return _getValues().containsKey(key);
  }

  List<String> get keys {
    return List.unmodifiable(_getValues().keys);
  }

  List<Object> get values {
    return List.unmodifiable(
      (_values ??= _file.read() ?? {}).values,
    );
  }

  Map<String, Object> _getValues() {
    return _values ??= _file.read() ?? {};
  }

  Object? getValue(String key) => _getValues()[key];

  bool valueEquals(Object? a, Object? b) {
    return const DeepCollectionEquality().equals(a, b);
  }

  bool? getBool(String key) => getValue(key) as bool?;
  int? getInt(String key) => getValue(key) as int?;
  double? getDouble(String key) => getValue(key) as double?;
  String? getString(String key) => getValue(key) as String?;
  List<Object?>? getList(String key) {
    return getValue(key) as List<Object?>?;
  }
  Map<Object, Object?>? getMap(String key) {
    return getValue(key) as Map<Object, Object?>?;
  }

  void setValue(String key, Object? value) async {
    if (value == null) {
      resetValue(key);
    }

    final values = Map.of(_getValues());
    final oldValue = values[key];
    values[key] = value!;
    if (oldValue == null) {
      _values = values;
      _file.write(values);
    } else if (!valueEquals(oldValue, value)) {
      _values = values;
      _file.write(values);
    }
  }
```

```
  void resetValue(String key) async {
    final values = Map.of(_getValues());
    if (values.remove(key) != null) {
      _values = values;
      _file.write(values);
    }
  }
}
```

Теперь перейдите к файлу `json_store.dart` каталога lib и добавьте в него одну строку с экспортом:

```
// lib - json_store.dart
export 'src/json_store.dart';
```

Работу реализованного хранилища будем наблюдать в несколько этапов. Сначала запишем данные и проверим их наличие по ключу, потом изменим пару записей и в конечном счете удалим хотя бы одну пару «ключ:значение». Для этого откройте файл `json_store.dart` каталога bin и замените имеющийся код на следующий:

```
// bin - json_store.dart
import 'package:json_store/json_store.dart';

void main(List<String> arguments) {
  final store = JSONStore('bin/store.json');
  store.setValue('strList', <String>['a', 'b', 'c']);
  store.setValue('int', 55);
  store.setValue('bool', true);
  store.setValue('double', 3.14);
  store.setValue('map', <String, int>{'a': 1, 'b': 2});
  store.setValue('str', '(づ｡•ᴗ•｡)づ♡');

  print(store.values);
  // [[a, b, c], 55, true, 3.14, {a: 1, b: 2}, (づ｡•ᴗ•｡)づ♡]
  print(store.keys);
  // [strList, int, bool, double, map, str]

  print(store.contains('strList')); // true
  print(store.getValue('strList')); // [a, b, c]
  print(store.getValue('int')); // 55
  print(store.getValue('bool')); // true
  print(store.getValue('double')); // 3.14
  print(store.getValue('map')); // {a: 1, b: 2}
  print(store.getValue('str')); // (づ｡•ᴗ•｡)づ♡
}
```

После запуска приложения в каталоге bin должен создаться файл `store.json` со следующим содержимым:

```
{
  "strList": [
    "a",
    "b",
    "c"
  ],
  "int": 55,
  "bool": true,
  "double": 3.14,
  "map": {
    "a": 1,
    "b": 2
  },
  "str": "(づ｡•ᴗ•｡)づ♡"
}
```

Перепишем код функции `main`, изменив пару значений в хранилище:

```
// bin - json_store.dart
void main(List<String> arguments) {
  final store = JSONStore('bin/store.json');

  store.setValue('strList', '-_-');
  store.setValue('double', 99);
  print(store.getValue('strList')); // -_-
  print(store.getValue('double')); // 99
}
```

После запуска приложения структура файла `store.json`, в котором содержатся данные хранилища, преобразится следующим образом:

```
{
  "strList": "-_-",
  "int": 55,
  "bool": true,
  "double": 99,
  "map": {
    "a": 1,
    "b": 2
  },
  "str": "(づ｡•ˇ•｡)づ♡"
}
```

Настало время удалить несколько записей:

```
// bin - json_store.dart
void main(List<String> arguments) {
  final store = JSONStore('bin/store.json');

  store.resetValue('map');
  store.resetValue('str');
  print(store.getValue('map')); // null
  print(store.getValue('str')); // null
}

// store.json
{
  "strList": "-_-",
  "int": 55,
  "bool": true,
  "double": 99
}
```

Когда вы точно знаете, значение какого типа данных хранится по ключу, можно воспользоваться следующим набором методов `JSONStore`:

```
  bool? getBool(String key) => getValue(key) as bool?;
  int? getInt(String key) => getValue(key) as int?;
  double? getDouble(String key) => getValue(key) as double?;
  String? getString(String key) => getValue(key) as String?;
  List<Object?>? getList(String key) {
    return getValue(key) as List<Object?>?;
  }
  Map<Object, Object?>? getMap(String key) {
    return getValue(key) as Map<Object, Object?>?;
  }
```

Что касается последних двух методов, то у их возвращаемого значения необходимо вызвать метод `cast` и указать, к каким типам привести элементы коллекции:

```
void main(List<String> arguments) {
  final store = JSONStore('bin/store.json');
  store.setValue('map', <String, int>{'a': 1, 'b': 2});
```

```
  store.setValue('strList', <String>['a', 'b', 'c']);
  store.setValue('intList', <int>[1, 2, 3]);

  var strList = store.getList(
                        'strList',
                      )?.cast<String>().toList();
  var intList = store.getList('intList')?.cast<int>().toList();
  var myMap = Map<String, int>.from(
    store.getMap('map')?.cast<String, int>() ?? {},
  );

  print(strList.runtimeType); // List<String>
  print(intList.runtimeType); // List<int>
  print(myMap.runtimeType); // _Map<String, int>

  print(strList); // [a, b, c]
  print(intList); // [1, 2, 3]
  print(myMap); // {a: 1, b: 2}
}
```

В противном случае для обращения к значению по ключу используется метод `getValue` с последующей проверкой того, к какому типу данных относится возвращаемое им значение.

Проект: игра «Крестики-нолики» v.3

В третью версию «Крестиков-ноликов» добавим возможность сохранения игровой сессии и ее загрузку при старте приложения, чтобы у игроков была возможность продолжить ранее остановленную игру. Откройте проект `tic_tac_toe` и добавьте в каталог src каталога lib следующие файлы (выделены жирным):

```
├── bin/
│   └── tic_tac_toe.dart
├── lib/
│   └── src/
│       ├── board.dart
│       ├── cell_type.dart
│       ├── game_state.dart
│       ├── game.dart
│       ├── i_game_loader.dart
│       ├── i_game_saver.dart
│       ├── json_game_state_worker.dart
│       └── player.dart
│
│   └── tic_tac_toe.dart
├── test/
└── pubspec.yaml
```

В файле `i_game_loader.dart` и `i_game_saver.dart` объявим интерфейсы, которые должен будет реализовать класс, отвечающий за сохранение и загрузку игровой сессии:

```
// lib/src/i_game_loader.dart
import 'game.dart';

abstract interface class GameLoader {
  Game? loadGame(String saveFilePath);
}

// lib/src/i_game_saver.dart
import 'game.dart';
```

```
abstract interface class GameSaver {
  void saveGame(Game currentGame, String saveFilePath);
}
```

Чтобы упростить код, реализуем сохранение игровой сессии в JSON-формате. Это потребует от нас внесения изменений в существующий код классов, а именно — добавления метода сериализации и фабричного именованного конструктора для десериализации.

Начнем эту эпопею с игрового поля. Откройте файл `board.dart` и добавьте в конец класса `Board` следующий код:

```
// lib/src/board.dart
import 'dart:io';

import 'cell_type.dart';

class Board {
 // код без изменений

  Map<String, dynamic> toJson() {
    return {
      'size': size,
      'cells': cells
          .map((List<Cell> row) =>
              row.map((Cell cell) => cell.toString()).toList())
          .toList(),
    };
  }

  factory Board.fromJson(Map<String, dynamic> json) {
    var board = Board(json['size']);
    board.cells = (json['cells'] as List)
        .map((row) => (row as List)
            .map((cell) => Cell.values.firstWhere(
                  (e) => e.toString() == cell,
                ))
            .toList())
        .cast<List<Cell>>()
        .toList();
    return board;
  }
}
```

Следующий на очереди — класс `Player`:

```
// lib/src/player.dart
import 'cell_type.dart';

class Player {
  // код без изменений

  Map<String, dynamic> toJson() {
    return {
      'cellType': cellType.toString(),
    };
  }

  factory Player.fromJson(Map<String, dynamic> json) {
    return Player(Cell.values.firstWhere(
      (e) => e.toString() == json['cellType'],
    ));
  }
}
```

Класс `Game` преобразится более существенно. В его конструктор будет передаваться экземпляр класса, отвечающий за сохранение и загрузку игровой сессии, приводящийся к интерфейсу `GameSaver`. Таким образом, мы осуществим сокрытие от класса `Game` всего функционала, предоставив доступ только к методу сохранения игровой сессии — `saveGame`. Другой добавляемый аргумент конструктора — `state` — позволит продолжать загруженную игру с того хода игрока, на котором произошло сохранение игровой сессии. Еще нам понадобится добавить метод обработки команды на сохранение, дополнительно внеся изменения в метод `play`. В отличие от класса `Player` и `Board`, в классе `Game` будет только метод сериализации:

```
// lib/src/game.dart
import 'dart:io';

import 'board.dart';
import 'cell_type.dart';
import 'game_state.dart';
import 'i_game_saver.dart';
import 'player.dart';

class Game {
  final Board board;
  final Player currentPlayer;
  GameState state;
  GameSaver saver;

  Game(
    this.board,
    this.currentPlayer,
    this.saver, {
    this.state = GameState.playing,
  });

  void updateState() {
    // код без изменений
  }

  bool saveCheck(String input) {
    if (input == "save") {
      print("Input file name:");
      String? fileName = stdin.readLineSync();
      if (fileName != null) {
        saver.saveGame(this, fileName);
      }
      return true;
    }
    return false;
  }

  void play() {
    while (state == GameState.playing) {
      board.printBoard();
      StringBuffer buffer = StringBuffer();
      buffer.write("${currentPlayer.symbol}'s turn. ");
      buffer.write("Enter row and column (e.g. 1 2): ");
      bool validInput = false;
      int? x, y;

      while (!validInput) {
        stdout.write(buffer.toString());
```

```
        String? input = stdin.readLineSync();
        if (input == null) {
          print("Invalid input. Please try again.");
          continue;
        }
        if (saveCheck(input)) {
          continue;
        }
    // далее код без изменений
  }

  Map<String, dynamic> toJson() {
    return {
      'board': board.toJson(),
      'currentPlayer': currentPlayer.toJson(),
      'gameState': state.name,
    };
  }
}
```

Далее реализуем класс, который отвечает за сохранение и загрузку игровой сессии и реализует интерфейсы `GameSaver` и `GameLoader`. Для этого откройте файл `json_game_state_worker.dart` и добавьте в него следующий код:

```
// lib/src/json_game_state_worker.dart
import 'dart:convert';
import 'dart:io';

import 'board.dart';
import 'game.dart';
import 'game_state.dart';
import 'i_game_loader.dart';
import 'i_game_saver.dart';
import 'player.dart';

class JsonGameStateWorker implements GameSaver, GameLoader {
  @override
  void saveGame(Game currentGame, String saveFilePath) {
    var encoder = JsonEncoder.withIndent('  ');
    File(saveFilePath).writeAsStringSync(encoder.convert(
      currentGame.toJson(),
    ));
  }

  @override
  Game? loadGame(String saveFilePath) {
    if (File(saveFilePath).existsSync()) {
      final gameStateString = File(saveFilePath).readAsStringSync();
      final gameStateJson = json.decode(gameStateString);
      final board = Board.fromJson(gameStateJson['board']);
      final currentPlayer = Player.fromJson(
        gameStateJson['currentPlayer'],
      );
      final state = GameState.values.firstWhere(
        (element) => element.name == gameStateJson['gameState'],
        orElse: () => GameState.playing,
      );
      return Game(board, currentPlayer, this, state: state);
    }
    return null;
  }
}
```

Далее добавьте в файл `tic_tac_toe.dart` каталога lib экспорт файла с только что реализованным классом:

```
// lib/tic_tac_toe.dart
export 'src/board.dart';
export 'src/game.dart';
export 'src/player.dart';
export 'src/cell_type.dart';
export 'src/json_game_state_worker.dart';
```

Последнее, что осталось, — внести изменения в функцию `main`, добавив меню с возможностью загрузки ранее сохраненной игры и запуском новой:

```
// bin/tic_tac_toe.dart
import 'dart:io';

import 'package:tic_tac_toe/tic_tac_toe.dart';

void main() {
  var saver = JsonGameStateWorker();
  int? size;

  print('1 - load game');
  print('2 - new game');
  print('q - quit');
  var input = stdin.readLineSync()!;
  switch (input) {
    case '1':
      Game? game;
      while (true) {
        print('Input file name:');
        String? fileName = stdin.readLineSync();
        if (fileName == null) {
          print('Invalid input');
          continue;
        }
        game = saver.loadGame(fileName);
        if (game != null) {
          break;
        } else {
          print('Game not found');
        }
      }
      game.play();
    case '2':
      while (true) {
        stdout.write('Enter the size of the board (3-9): ');
        size = int.tryParse(stdin.readLineSync()!);
        size ??= 3;
        if (size < 3 || size > 9) {
          continue;
        }
        break;
      }

      Board board = Board(size);
      Player player = Player(Cell.cross);
      Game game = Game(board, player, saver);
      game.play();
    case _:
      print('Exit...');
      break;
  }
}
```

Запустите игру, сделав пару ходов за каждую из сторон, после чего сохраните игровую сессию в файле `myGame`. При вводе символа `q` одним из игроков можно закончить игру, выйдя в главное меню, что позволит загрузить записанную ранее игровую сессию:

```
1 - load game
2 - new game
q - quit
2
Enter the size of the board (3-9): 4
  1 2 3 4
1 . . . .
2 . . . .
3 . . . .
4 . . . .
X's turn. Enter row and column (e.g. 1 2): 2 3
  1 2 3 4
1 . . . .
2 . . X .
3 . . . .
4 . . . .
O's turn. Enter row and column (e.g. 1 2): 1 2
  1 2 3 4
1 . O . .
2 . . X .
3 . . . .
4 . . . .
X's turn. Enter row and column (e.g. 1 2): 4 3
  1 2 3 4
1 . O . .
2 . . X .
3 . . . .
4 . . X .
O's turn. Enter row and column (e.g. 1 2): save // сохранение
Input file name:
myGame
O's turn. Enter row and column (e.g. 1 2): q // выход в главное меню
  1 2 3 4
1 . O . .
2 . . X .
3 . . . .
4 . . X .

************************ загрузка файла *****************
1 - load game
2 - new game
q - quit
1
Input file name:
myGame
  1 2 3 4
1 . O . .
2 . . X .
3 . . . .
4 . . X .
O's turn. Enter row and column (e.g. 1 2): 4 4
  1 2 3 4
1 . O . .
2 . . X .
3 . . . .
4 . . X O
X's turn. Enter row and column (e.g. 1 2):
```

Ниже приведено содержимое записанного файла:

```
// myGame
{
  "board": {
    "size": 4,
    "cells": [
      [
        "Cell.empty",
        "Cell.nought",
        "Cell.empty",
        "Cell.empty"
      ],
      [
        "Cell.empty",
        "Cell.empty",
        "Cell.cross",
        "Cell.empty"
      ],
      [
        "Cell.empty",
        "Cell.empty",
        "Cell.empty",
        "Cell.empty"
      ],
      [
        "Cell.empty",
        "Cell.empty",
        "Cell.cross",
        "Cell.empty"
      ]
    ]
  },
  "currentPlayer": {
    "cellType": "Cell.nought"
  },
  "gameState": "playing"
}
```

В конце следующей главы мы полностью изменим приложение, добавив возможность проводить игровые партии в выделенных комнатах на сервере. А пока предлагаю выполнить следующие задания, внеся изменения в кодовую базу проекта.

1. В данный момент при запуске приложения из IDE файл с игровой сессией сохраняется в корневой каталог проекта, что неправильно. Измените код, чтобы файлы сохранялись в каталог data (если такого нет, то он создается автоматически) с расширением `.json`.
2. Если пользователь забыл имя файла с сохранением, то будет мучиться до тех пор, пока не вспомнит его или не закроет приложение нажатием клавиш Ctrl+C, после чего запустит его снова. Исправьте данный момент.
3. Держать в голове название сохраненных файлов довольно утомительно. Предоставьте пользователю возможность выбирать из списка, какой файл с записанной игровой сессией должен быть загружен приложением.
4. Покройте добавленный и измененный код приложения тестами.
5. Придумайте свой формат хранения игровой сессии в текстовом файле и реализуйте его.

Резюме

В данной главе мы рассмотрели, какими способами можно осуществлять компиляцию разрабатываемого приложения и как его конфигурировать в момент запуска через терминал. Кроме того, мы поговорили почти обо всех основных возможностях Dart по работе с файлами и каталогами. Почему почти? Потому что Dart может манипулировать данными и в HTML-файлах, но с появлением Flutter для Web эти его возможности теряют актуальность. Да и признаемся честно: не так уж много компаний горят желанием использовать его в этих целях, когда JavaScript и Flutter предоставляют куда больший набор библиотек и возможностей.

Дополнительно мы рассмотрели такой формат представления данных, как JSON, и способы работы с ним. Он часто будет встречаться на вашем пути, и чем раньше у вас получится подружиться с ним, тем лучше.

Использовать библиотеки для генерации кода довольно удобно, но не спешите применять их везде, так как они тянут в ваш проект ряд зависимостей, с которыми впоследствии придется считаться, особенно если одну из них прекратит поддерживать ее автор, да и сообществу до нее не будет никакого дела.

Вопросы для самопроверки

1. Какие флаги для компиляции приложения вы знаете? Назовите их различия, достоинства и недостатки.
2. Каковы способы конфигурации приложения при его запуске средствами терминала?
3. Какой класс в Dart позволяет работать с файлами? Перечислите его основные методы.
4. Каковы режимы работы с файлами? В чем их различия?
5. Какой класс лучше использовать при необходимости прочитать/загрузить файл большого размера?
6. Как проверить существование файла в системе?
7. Как получить путь до каталога запускаемого приложения?
8. Как явным образом создать файл? За что отвечает флаг `recursive`?
9. Какой класс в Dart позволяет работать с каталогами? Перечислите его основные методы.
10. Что такое JSON? Для чего и где он используется?
11. Что такое сериализация и десериализация?
12. Для чего можно использовать библиотеку `json_serializable`? Каковы ее ограничения?
13. Стоит ли всегда использовать библиотеки для генерации кода? Почему?