
Flutter™
Быстрый старт
Flutter разработчика
Пошаговое пособие
разработчика кросс-платформенных
мобильных приложении
Андрей Алеев
Flutter and the related logo are trademarks of Google LLC.
We are not endorsed by or affiliated with Google LLC.

Андрей Алеев

Быстрый старт Flutter-разработчика

http://www.litres.ru/pages/biblio_book/?art=48781701
ISBN 9785005087973

Аннотация

В этой книге даны необходимые элементы, база, которую нужно знать Flutter-разработчику, чтобы писать кросс-платформенные мобильные приложения под Android и iOS на языке Dart. Все это представлено в наглядной форме, на практических примерах, в формате уроков. После их освоения вы сможете именовать себя Flutter-разработчиком.Flutter and the related logo are trademarks of Google LLC. We are not endorsed by or affiliated with Google LLC.

Содержание

Введение 5
Как работать с этой книгой 7
Урок 1. Запускаем Flutter 11
Кроссплатформенная мобильная разработка 11
Почему Flutter? 13
Настраиваем рабочее окружение 15
Запускаем Hello World! На Android 17
Запускаем Hello World на iOS 25
Урок 2. Язык программирования Dart 29
Введение 29
Переменные, типы и область видимости 31
Видимость 32
Типы 33
final и const 38
Функции 39
Конец ознакомительного фрагмента. 41

Быстрый старт Flutter-разработчика

Андрей Алеев

ISBN 978-5-0050-8797-3
Создано в интеллектуальной издательской системе Ridero

Введение

Начиная с 2015 года, с момента анонсирования Flutter SDK, популярность этой платформы и языка Dart растет неукоснительно. На популярных профильных ресурсах нарастает количество статей по данной тематике, а многие компании выпускают в магазины приложения, созданные с помощью Flutter™.

Цель данной книги – научить вас создавать кроссплатформенные мобильные приложения под Android и iOS на Flutter. На практических примерах мы разберем основы языка Dart и базовые принципы построения Flutter-приложений.

Книга будет интересна нативным мобильным разработчикам, которые уже занимаются разработкой приложений, а также всем, кто желает начать писать кроссплатформенные мобильные приложения и познакомиться с языком Dart. Необходимы только базовые знания по программированию. Опыт front-end-разработки придется очень кстати – с ним материал курса будет освоить гораздо проще. Тем не менее, иметь его совсем не обязательно, тем более что после освоения этой книги вы будете на один большой шаг ближе к тому, чтобы именоваться мастером front-end-девелопмента.

Ученые из Оксфордского университета выяснили, что

всего лишь 400 слов покрывают 75% всех английских текстов. Это означает, что со словарным запасом в 400 самых используемых слов вы в трех случаях из четырех будете знать, о чем идет речь в любом тексте. Аналогичным образом написана данная книга: она не претендует на звание учебника или полного справочника платформы Flutter и языка Dart. Мы не будем разбирать по очереди каждый из виджетов в библиотеке material, не будем заучивать все ключевые слова языка Dart. Наоборот, здесь даны самые необходимые элементы, минимум, который надо знать Flutter-разработчику в продакшн, то есть в приложении к решению настоящих задач: *созданию мобильных приложений для реального мира*.

Как работать с этой книгой

Лучше всего усваивается информация, полученная эмпирическим путем. Поэтому ожидается, что вы будете не просто пассивно читать эту книгу, а по каждому уроку напишете код и запустите приложение на двух платформах – Android и iOS.

В идеале, постарайтесь написать свое приложение, которое будет, к примеру, загружать фотки котиков из сети или выполнять более утилитарную задачу, пусть калькулятор. На ваш вкус. В этой книге мы будем разбирать два примера – сначала создадим простой счетчик, а затем более сложный – загрузка прогноза погоды с сайта openweathermap.org. Если вы захотите написать такое же приложение, вам потребуется API KEY с их сайта, а также API KEY Google Maps. Помимо этого, желательно иметь опыт работы с Git, Android Studio, Gradle.

Всего в книге 10 глав-уроков, первые уроки более простые, последние – более сложные, и для них, возможно, потребуется больше времени. Помогать вам будет уже написанный и работающий код в репозитории проекта – https://github.com/acinonyxjubatus/flyflutter_fast_start – FlyFlutter Fast Start на гитхабе, там для каждого урока выделена своя ветка. Старайтесь не просто копировать оттуда код, а вдумчиво писать его, только лишь сверяясь с ко-

дом на гитхабе. Ниже вкратце приведено описание уроков, а также указаны ссылки на соответствующие ветки репозитория.

Урок 1. Запускаем Flutter [ветка *lesson_1_hello_world*]

Научимся запускать проект на Flutter под Android и iOS, а также совершать простейшие манипуляции с виджетами. Помимо этого, узнаем чем может быть полезен Flutter и когда на нем можно создавать приложения.

Урок 2. Язык программирования Dart

Обзорно пройдемся по основным возможностям и правилам языка Dart

Урок 3. StatelessWidget и StatefulWidget [ветки *lesson_3_1_stateless_widget, lesson_3_1_stateful_widget*]

Научимся создавать Stateless и StatefulWidget-ы. Узнаем про состояния виджетов, попробуем ими манипулировать. Также узнаем, как декорировать и выравнивать виджеты.

Урок 4. Создание списка элементов [ветка *lesson_4_listivew*]

Познакомимся с ListView, узнаем какие есть способы его создания. Полученные знания применим для создания списка с прогнозами погоды.

Урок 5. Загрузка данных с сервера [ветка *lesson_5_http*]

Узнаем как можно выполнить асинхронную работу во Flutter. Сделаем запрос на сервер, получим, распарсим и покажем полученную информацию на клиенте. Таким об-

разом, создадим полноценное клиент-серверное приложение.

Урок 6. Inherited Widgets, Elements, Keys [ветка *lesson_6_inherited*]

Узнаем, что такое Inherited Widget, а также на примере посмотрим как он работает. Разоберемся с тем, что такое Element-ы и как они работают. Помимо этого, мы познакомимся с ключами Keys и узнаем когда и как их нужно использовать.

Урок 7. Навигация между экранами, Работа с Google Maps [ветка *lesson_7_navigation_maps*]

Научимся переключать экраны с помощью Navigator-a. Сможем подключить и показать карты от Google Maps в приложении, а также подключим дополнительный необходимый в примере пакет timezone.

Урок 8. SQLite, Clean Architecture [ветка *lesson_8_sqlite_clean_architecture*]

Сумеем подключить SQLite и сохранить данные в локальной базе данных, а также прочесть их. Убедимся, что во Flutter тоже можно и нужно писать чистый код и напишем свою реализацию паттерна Repository.

Урок 9. BLoC, Streams [ветки *lesson_9_bloc, lesson_9_1_counter_bloc*]

Узнаем, что такое BLoC, чем он полезен и как использовать библиотеку bloc. Все это применим на практике: мы

произведем значительный рефакторинг приложения погоды, придав коду приличествующий вид – повысим читаемость и поддерживаемость.

Урок 10. DI, Тесты [ветки *lesson_10_di_tests, lesson_9_1_counter_bloc*]

Освоим технику инверсии зависимостей применительно к Flutter разработке. На практическом примере реализуем паттерн Dependency Injection во Flutter в примере приложения погоды. Затем узнаем, какие бывают тесты. Напишем unit-тесты, widget (UI-тесты) и интеграционные тесты для приложения с погодой.

Урок 1. Запускаем Flutter

В этой главе:

– Кроссплатформенная мобильная разработка
– Почему Flutter?
– Настраиваем рабочее окружение
– Запускаем Hello World на Android
– Запускаем Hello World на iOS

Кроссплатформенная мобильная разработка

Для начала несколько слов о том, что такое Flutter и зачем он нам нужен. Если вы знаете ответ на вопрос, что такое кроссплатформенная разработка и Flutter, листайте дальше к пункту 3 этой главы: «Настройка рабочего окружения».

Так вышло, что на сегодняшний день в мире мобильных устройств лидируют 2 платформы – iOS от Apple и Google Android. Представьте, что вам прямо сейчас надо написать мобильное приложение под обе операционные системы. Вам нужно нанять, условно, по 1—3 программиста на каждую платформу. Или по 5, или по 7, в зависимости от сложности проекта.

Возьмем число 5 на платформу – оптимальное, на мой

взгляд, количество для проекта средней сложности. Это означает 10 программистов в сумме. Из них статистически будет 2—4 очень хороших, сильных программиста, 2—4 слабеньких и 2—4 средних по уровню. Если же язык программирования один и кодовая база одна, значит, можно взять из этих же 10 программистов 5 лучших. Конечно, останется 5 программистов не у дел, но это возможность переместить их на другие участки работы или дополнительный стимул расти им профессионально. Иными словами, сузив скоуп работ до одной кодовой базы, можно одновременно уменьшить расходы на разработку и увеличить качество. Конечно, это все теория. На практике большинство выбирают нативную разработку, и зачастую оправдано, поскольку только она дает максимальное качество конечного продукта. Но зачастую – не значит всегда. Рассмотрим, когда и как можно применить Flutter.

Почему Flutter?

Если Вы думаете, стоит ли Вам браться за кросс-платформу и конкретно за Flutter, ответьте себе на вопрос: зачем нам нужно это приложение, какие бизнес-цели мы с помощью него решаем? Сравните свой ответ с двумя абзацами ниже и решите к какому относится ваше приложение в большей степени.

Для начала определим, в каких случаях Flutter не очень хорошо подходит. Если кратко, то это все кейсы, когда приложение *представляет собой конечный продукт* и будет конкурировать с другими такими продуктами в магазине приложений за топовые позиции. Например, это может быть новая Angry Birds, рисовалка, читалка, фитнес-приложение. Вам нужна будет максимальная скорость, точность и плавность при работе приложения, и это все на сегодняшний день дает только нативное приложение. Также следует выделить категорию приложений, в которых планируется активно использовать встроенные в устройства датчики, такие как Bluetooth, гироскопы, камеру. Это конечно, не значит, что Flutter нельзя использовать в перечисленных случаях. Но высока вероятность, что вам так или иначе придется писать нативный код и/или костыли.

С другой стороны, существует множество кейсов, когда реальный бизнес желает получить мобильное приложение,

которое будет помогать им в реализации бизнес-процессов и/или дополнять их, но без фанатичной погони за самым модным UI и супер-быстродействием. В качестве примера можно привести программы лояльности, мобильное рабочее место для сотрудников, интернет-магазин, а также многие другие, где приложение будет *обслуживать реальный бизнес-процесс*.

Резюмируя, небольшие приложения с оффлайновым бизнесом можно и нужно создавать на Flutter, а сам framework рекомендуется к изучению всем мобильным разработчикам.

Настраиваем рабочее окружение

Теперь, когда мы разобрались, в каких случаях мы можем использовать Flutter, давайте уже научимся им пользоваться!

Для начала установим Flutter SDK. Скачайте архив с SDK с официального сайта (https://flutter.dev/docs/get-started/install). Выберите вашу платформу (Windows, Mac, Linux) и следуйте инструкции.

После распаковки архива добавьте в PATH Flutter/bin

export PATH=«$PATH:`pwd`/flutter/bin» // Mac

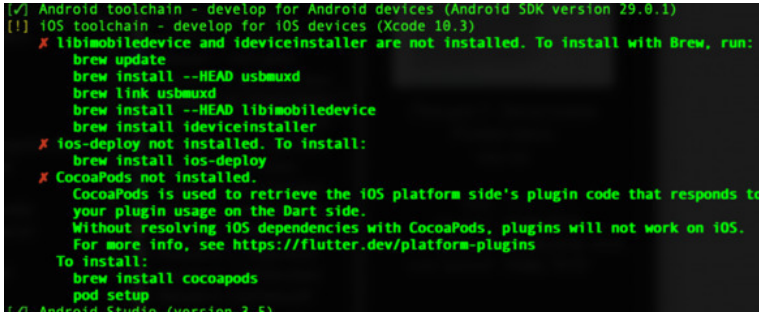
Здесь может потребоваться перезапустить компьютер.

После установки в командной строке запустите команду

flutter doctor

и убедитесь, что у вас все установлено корректно.

Если планируете собирать и тестировать под iOS, то необходимо установить и обновить Xcode и соответствующие пакеты с помощью brew, следуя подсказке в ответе flutter doctor, а также следовать инструкции для macos https://flutter.dev/docs/get-started/install/macos)



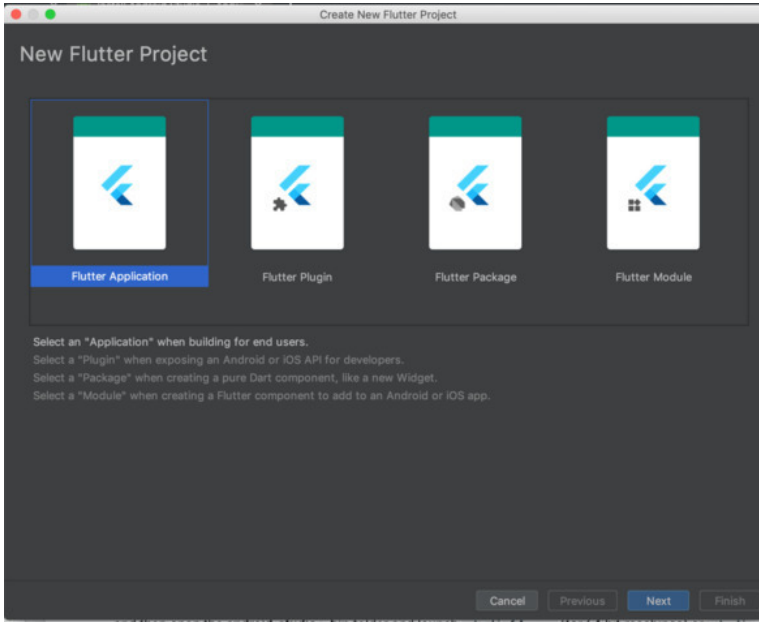
Результаты команды flutter doctor с ошибками

Если планируете тестировать на Android-устройстве, то используйте Android Studio. Если у вас нет Android Studio, следуйте инструкции по установке (https://developer.android.com/studio/install), чтобы установить ее.

Запускаем Hello World! На Android

Итак, приступим к созданию первого приложения на Flutter. Для этого курса вы также можете использовать Android Studio, XCode или VS Code – как вам удобно. Мы будем рассматривать на примере Android Studio.

Запустите Android Studio и выберите *Start a new Flutter project.*



Интерфейс создания нового проекта

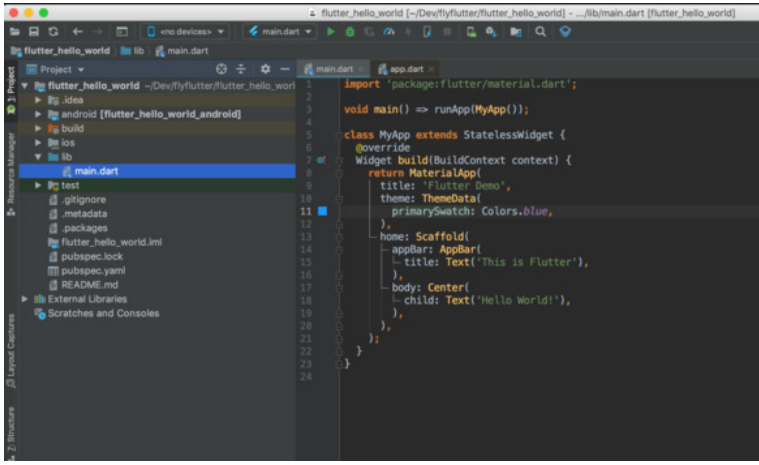
Выберите **Flutter Application**
Заполните имя *flutter_hello_world* в поле **Project Name**
company domain – **flyflutter.ru** – и жмем Finish.

После запуска мы сразу видим открытый файл **main. dart** В нем – видим строчку

```
void main () => runApp (MyApp ());
```

это начальная точка приложения. Функция **main ()** – это стартовая точка всех приложений на языке Dart. В ней мы здесь вызываем конструктор класса **MyApp**, который наследуется от **StatelessWidget** – это тип UI компонента – виджета. Подробнее про язык Dart мы поговорим во второй лекции, а про виджеты – в третьей.

Итак, слева мы видим дерево проекта, справа – редактор.



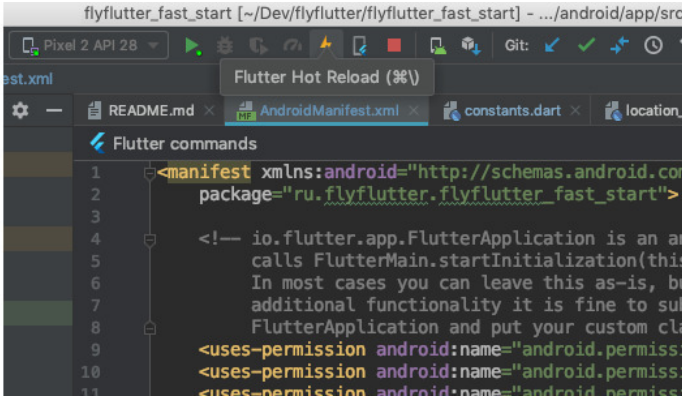
Код main. dart только что созданного проекта

Весь общий для Android и iOS код находится в папке lib. Сейчас у нас там только файл main. dart

Android Studio сгенерировала простую логики инкрементирования счетчика, мы ее пока удалим, чтобы она нас не путала, и заменим на более простой вариант

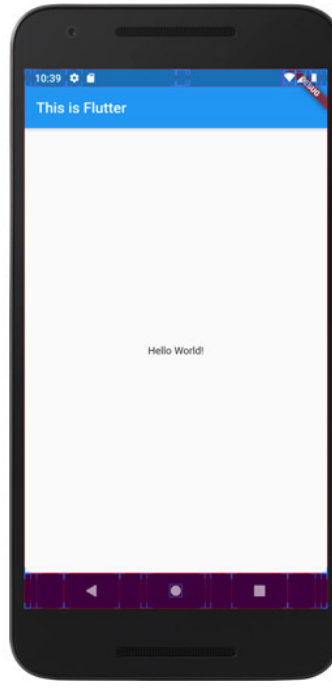
```dart
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: Scaffold(
        appBar: AppBar(
          title: Text('This is Flutter'),
        ),
        body: Center(
          child: Text('Hello World!'),
        ),
      ),
    );
  }
}
```

Жмите на иконку молнии – **Hot Reload** – для применения изменений.



Надо отметить, что Hot Reload во Flutter работает действительно быстро и значительно сокращает время разработки.

Ура, на экране вы должны увидеть «Привет, Мир!».



«Привет, Мир!» на эмуляторе

Рассмотрим код подробнее. Как уже говорилось выше, MyApp наследуется от StatelessWidget, это неизменяемый UI компонент-виджет. Вообще, все во Flutter – это виджеты, и приложение тоже. В виджете мы переопределяем метод build, в котором указывается, что и как отрисовать.

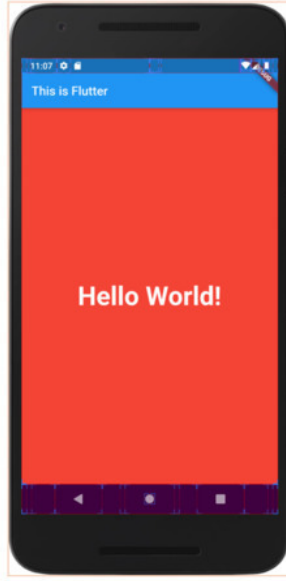
В нашем примере мы возвращаем объект MaterialApp, который создаем посредством конструктора. А в конструктор

передаем название, тему и виджет home, которому назначаем Scaffold – скелет приложения, который в свою очередь содержит appBar и body. Здесь уместна аналогия с HTML, где также есть тэги <title> и <body>.

Давайте немного увеличим текст и поиграем цветами:

```
home: Scaffold(
  backgroundColor: Colors.red,
  appBar: AppBar(
    title: Text('This is Flutter'),
  ),
  body: Center(
    child: Text('Hello World!',
        style: TextStyle(
          fontSize: 42.0, // делаем текст большим
          fontWeight: FontWeight.bold,  // жирным
          color: Colors.white, // белым
        )
    ),
  ),
),
```

Виджету Scaffold мы задали красный фон, а виджету текста применили стиль, чтобы сделать его больше и заметнее.

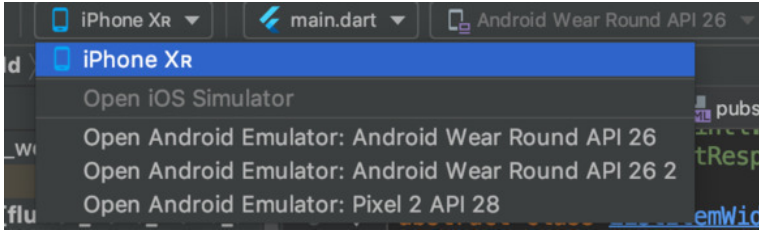


Привет, Мир! на Андроид

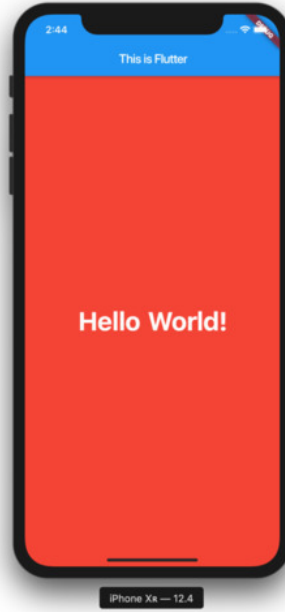
Преимущество Flutter в том, что вся логика работы с внешним видом приложения (UI) прописывается в коде на том же языке, что и бизнес-логика – на dart. Нет необходимости залезать в папку с ресурсами и редактировать xml верстку.

Запускаем Hello World на iOS

Мы же пишем кроссплатформенный код! Давайте запустим созданное приложение на iOS-девайсе. Для этого просто выберите подключенный iOS-девайс или эмулятор в dropdown-списке и нажмите «Запустить».



Выбор эмулятора iOS



Привет, Мир! на iOS

Экран выглядит потрясающе, однако вверху экрана мешается ненужная иконка debug, да и иконка приложения сейчас никакая. Исправим это.

Для того, чтобы убрать ленточку debug, в добавьте в MaterialApp флаг *debugShowCheckedModeBanner* со значением false

return MaterialApp (
debugShowCheckedModeBanner: false,

Чтобы поменять иконку, нужно добавить в pubspec. yaml пакет

dev_dependencies:
flutter_launcher_icons: ^0.7.4

Этот пакет значительно упростит нам добавление иконки для двух платформ сразу. Добавим теперь в корне проекта папку assets с иконкой, а также пропишем путь к иконке

flutter_icons:
android: «launcher_icon»
ios: true
image_path: «assets/icons/flyflutter_ic_512.webp»

не забудьте сказать flutter, чтобы смотрел папочку assets

flutter:

uses-material-design: true
assets:
– assets/
– assets/icons/

После этого для генерации иконок запустите в терминале

команды

flutter pub get
flutter pub run flutter_launcher_icons: main

Чтобы поменять лейбл (название иконки) приложения:

Для Android – найдите манифест в android/app/src/main/ AndroidManifest. xml и добавьте в тег application строку

android: label=«FlyFlutter»

Для iOS же зайдите в Info.plist по пути ios>runner/Info.plist и для ключа укажите имя **CFBundleName**

<key> CFBundleName </key>
<string> FlyFlutter </string>

Готово. Запустите снова для проверки.

Урок 2. Язык программирования Dart

В этой главе:

– Переменные, типы и область видимости
– Функции
– Конструкторы
– Наследование
– Примеси (mixin)
– Callable классы
– Дженерики
– Асинхронные функции
– Исключения
– Использование библиотек
– Компиляция

Введение

Приложения под Flutter пишутся на языке Dart. Даже сам фреймворк написан на нем. Dart – это высокоуровневый объектно-ориентированный язык программирования общего назначения с открытым исходным кодом. Был разработан в Google. Испытал влияние C, Javascript, C#, Java. В нем также как и в Java и C# присутствует garbage collector. Язык поддерживает интерфейсы, примеси (англ. Mixin), абстракт-

ные классы, дженерики и статическую типизацию.

Dart был представлен публике в 2011 году авторами Ларсом Барком (Lars Bark) и Каспером Лундом (Kasper Lund). Релиз версии 1.0 состоялся в 2013 году, а версии 2.0 в 2018

Примечание: Здесь и далее в этом курсе мы рассматриваем Dart версии 2

Все приложения на Dart, как и на C и в Java, имеют точку входа в функции main ()

```
void main() {
  print('Hello, World!');
}
```

В случае, если необходимо запустить программу на Dart из командной строки, то можно использовать параметризованную main:

```
void main(List<String> args) {
  print(args);
}
```

Переменные, типы и область видимости

Dart типобезопасный язык. В нем используется как статическая типизация на этапе компиляции, так и динамическая проверка во времени исполнения (runtime) программы. Несмотря на наличие статической типизации, указывать тип переменной необязательно. Например, все объявления и инициализации ниже корректные:

```
var name = 'Dart';
var year = 2011;
String author;
author = "Lars Bark";
List<Foo> myList = <Foo>[];
List<Foo> oldList = new List();
```

Примечание: В Dart 2 ключевое слово **new** стало необязательным

Видимость

По умолчанию, все переменные имеют публичную область видимости. Таких привычных для Java-программистов ключевых слов, как private, protected и public в Dart нет.

Однако если добавить нижнее подчеркивание [_] к имени переменной, такая переменная будет иметь область видимости библиотеки, в которой она находится.

Типы

Все объекты в Dart наследуются от базового типа Object. Это аналог Object в Java. В нем также есть метод hasCode () и аналог equals, который заменяет оператор сравнения ==

Так же в классе Object присутствует метод toString ()

Встроенные типы включают:

– Числовые (num и его наследники int и double)
– Строковые (strings)
– Булевы (Booleans)
– Списки, или массивы (list)
– Сеты (set)
– Мапы (map)
– Руны (for expressing Unicode characters in a string)
– Символы (symbols)

int – Целочисленные переменные. На виртуальной машине Dart диапазон составляет от -2^{63} до 2^{63}—1

Примечание: При компиляции в JavaScript диапазон int-а -2^{53} до 2^{53}—1

double – 64-битные числа с плавающей запятой

И **int** и **double** наследуются от типа **num**

String

Строковые переменные в Dart представляют собой последовательности из UTF-16 символов. Для инициализации можно использовать как двойные, так и одинарные кавычки:

```
var s1 = 'Строка в одинарных кавычках';
String s2 = "Строка в двойных кавычках";
```

Значения переменных можно использовать в строках с помощью конструкции $ {выражение}

```
var a = 2;
var b = 2;
var s = '$a + $b = ${a+b}';
// получим «2+2=4»
```

bool

Для создания булевых переменных в Dart существует ключевое слово bool. При инициализации можно использовать литералы true и false. То есть, инициализация bool b = 0; – некорректна, правильно

```
bool b = true;
```

List

Списки – это коллекции проиндексированных объектов. Примеры объявления и инициализации списков:

```
List<int> list1 = new List();
List<int> list2 = List();
var list3 = [1, 2, 3];
```

Для инициализации в Dart 2.3 добавлен спред оператор – троеточие – с помощью него можно добавить в список множество значений:

```
var list = [1, 2, 3];
var list2 = [0, ...list];
```

Sets

Сеты – это неупорядоченные наборы уникальных элементов. В Dart для того, чтобы создать сет, нужно использовать фигурные скобки для непустого набора и фигурные скобки в сочетании с угловыми и типом объектов для пустого:

```
var colors = {'red', 'green', 'blue'}; // сразу инициализируем
сет
var colors = <String>{}; // объявляем пустой сет
Set<String> colors = {}; // тоже корректно
```

Maps

Мапы – это наборы данных в формате ключ-значение. Ключами, как и значениями, могут быть объекты любых типов. Каждый ключ является уникальным, значения могут быть разными, а могут дублироваться. Посмотрим на примере:

```
var ballGames = {
  'baseball': 'club',
  'basketball': 'hands',
  'football': 'foots'
};
```

Альтернативные способы инициализации

```
var ballGames = Map();
ballGames['baseball'] = 'club';
...

var ballGames = Map();
ballGames[2] = 'hands';
```

Runes

Dart поддерживает руны – спецсимволы юникод. Используйте, если хотите добавить смайлики. Попробуйте запустить в dartpad

```
Runes input = new Runes('\u{1f60e}');
print(new String.fromCharCodes(input));
```

final и const

В языке также присутствуют ключевые слова final и const.

Если переменную не планируется изменять, то следует задать ей модификатор final перед типом или словом var. Такая переменная может быть проинициализирована единожды. Переменные const неявно считаются final. Такие переменные используются для задания констант на этапе компиляции.

Функции

В Dart даже функции являются объектами. Это значит, что функции можно назначать переменным и передавать в качестве аргументов в другие функции. Тип возвращаемого значения указывается перед именем функции. Делать это необязательно, хотя и рекомендуется:

```
int doubleIt(int value) {
  return value * 2;
}

doubleIt(value) { // корректно
  return value * 2;
}
```

Поскольку эта функция содержит всего одно выражение, ее можно укоротить до одной строчки:

```
int doubleIt(value) => value * 2;
```

Оператор => -это сокращение фигурных скобок и слова return.

Опциональные параметры

При объявлении функции мы можем в ее сигнатуре указать значения по умолчанию. Например, нам понадобится вызывать какую-то функцию много раз с одним и тем же параметром, но при этом необходимо сохранить гибкость. В таком случае, при вызове функции с параметром по умолчанию его (этот параметр) можно не указывать.

В Dart существует два типа опциональных параметров: позиционные и именованные. Рассмотрим их подробнее.

Конец ознакомительного фрагмента.