|  |  |  |
| --- | --- | --- |
| Задание №4: Эссе  Технология кроссплатформеннной мобильной разработки: **Flutter** |  | **Выполнил:** Зимин Илья  **Группа:** ПМИ-2  **Дата сдачи:** 09.04.2020 |

**Flutter**

*Flutter* - это инструментарий пользовательского интерфейса Google для создания красивых, скомпилированных в естественном виде приложений для мобильных устройств, веб-сайтов и настольных ПК из единой базы кода.

*Flutter* *SDK* - бесплатное кроссплатформенное средство от Google с открытым исходным кодом для создания мобильных приложений. Он используется для разработки приложений под Android и iOS, а также это пока единственный способ разработки приложений под Google Fuchsia. Некоторые характеристики Flutter SDK приведены в таблице 1.

Таблица 1 – Характеристики Flutter SDK

|  |  |
| --- | --- |
| **Разработчик:** | Google |
| **Дата выхода:** | май 2017 г. |
| **Написана на:** | C, C++, Dart, Skia |
| **Последняя версия:** | 1.2.13+hotfix.5 |
| **Написано на:** | Си, C++, Dart |
| **Платформы:** | Android, iOS, Google Fuchsia, Веб-платформа, Windows, macOS, Linux |

**Базовая идея, возможности и особенности Flutter**

Основная идея Flutter вращается вокруг виджетов (Widgets). Весь пользовательский интерфейс состоит из объединения различных виджетов, каждый из которых определяет структурный элемент (например, кнопку или меню), стилистический элемент (например, шрифт или цветовую схему), аспект макета, (например, отступы) и т.д. Приложение построено из виджетов как конструктор. Flatter не использует виджеты OEM, но предоставляет возможность создавать собственные и включает набор готовых виджетов, которые применимы как для Android (material design), так и для IOS (Cupertino).

Google утверждает, что выбирая Flutter, мы получаем:

1. Быструю разработку (Fast Development),
2. Выразительный и гибкий интерфейс (Expressive and Flexible UI),
3. Нативную производительность (Native Performance).

Высокая производительность приложения и скорость разработки достигается за счет нескольких техник:

* В отличии от многих известных на сегодняшний день мобильных платформ, Flutter не использует JavaScript ни в каком виде. В качестве языка программирования для Flutter выбрали ***Dart***, который компилируется в бинарный код, за счет чего достигается скорость выполнения операций сравнимая с Objective-C, Swift, Java, или Kotlin,
* Flutter не использует нативные компоненты, опять же, ни в каком виде, так что не приходится писать никаких прослоек для коммуникации с ними. Вместо этого, подобно игровым движкам, он отрисовывает весь интерфейс самостоятельно. Кнопки, текст, медиа-элементы, фон – все это отрисовывается внутри графического движка в самом Flutter. После вышесказанного стоит отметить, что “Hello World” приложение на Flutter занимает совсем немного места: iOS ≈ 2.5Mb и Android ≈ 4Mb,
* Для построения UI во Flutter используется декларативный подход, вдохновленный веб-фреймворком ReactJS, на основе виджетов (в мире веба именуемых компонентами). Для еще большего прироста в скорости работы интерфейса виджеты перерисовываются по необходимости — только когда в них что-то изменилось (подобно тому как это делает Virtual DOM в мире веб-фронтенда),
* В дополнение ко всему, в фреймворк встроен Hot-reload, такой привычный для веба, и до сих пор отсутствовавший в нативных платформах.

**Архитектура Flutter**

* Платформа Dart
* Движок Flutter
* Библиотека Foundation
* Наборы виджетов

**Dart**

Как было сказано выше, приложения Flutter пишутся на языке Dart, используется Dart – язык программирования, созданный Google. Dart позиционируется в качестве замены/альтернативы JavaScript. Flutter работает в виртуальной машине Dart с JIT-компилятором. Из-за ограничений на динамическое выполнение кода в App Store, под iOS Flutter использует AOT-компиляцию.

Одно из главных преимуществ платформы Dart — «горячая перезагрузка» (Hot-reload), когда изменение исходного кода применяется сразу в работающем приложении без необходимости его перезапуска.

**Движок Flutter**

Движок написан преимущественно на C++, он поддерживает низкоуровневый рендеринг с помощью графической библиотеки Google Skia. А также имеет возможность взаимодействовать с платформозависимыми SDK под Android и iOS.

**Библиотека Foundation**

Эта библиотека, написанная на языке Dart, содержит основные классы и методы для создания приложений Flutter и взаимодействия с движком Flutter.

**Виджеты**

Как уже упоминалось, дизайн пользовательского интерфейса приложений Flutter обычно включает в себя использование и/или создание различных виджетов. Виджет Flutter представляет собой неизменяемое описание какой-либо части пользовательского интерфейса. Все графические объекты, включая текст, формы и анимацию, создаются с помощью виджетов. Комбинированием простых виджетов создаются комплексные виджеты. Однако создавать приложения Flutter можно и без виджетов, напрямую вызывая методы библиотеки Foundation для работы с канвой.

Фреймворк Flutter состоит из двух наборов виджетов, соответствующих конкретным описаниям дизайна: Material Design от Google и Cupertino для имитации дизайна приложений в Apple iOS.

Теперь рассмотрим ***архитектуру высокого уровня Flutter***, которая представлена на рисунке 1.

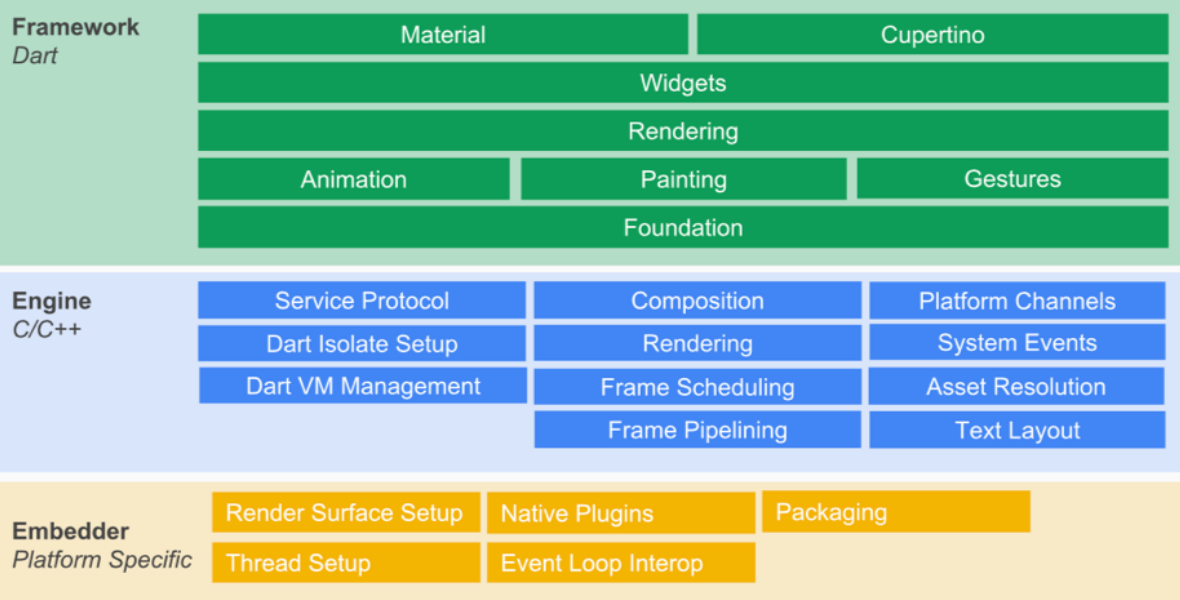


Рисунок 1 – архитектура высокого уровня Flutter

Когда мы пишем приложение Flutter, используя Dart, мы остаемся на уровне Flutter Framework (выделено зеленым цветом), который взаимодействует с Flutter Engine (синим цветом) через слой абстракции, называемый Window. Этот уровень абстракции предоставляет ряд API для косвенного взаимодействия с устройством.

Также через этот уровень абстракции Flutter Engine уведомляет Flutter Framework, когда:

* событие, представляющее интерес, происходит на уровне устройства (изменение ориентации, изменение настроек, проблема с памятью, состояние работы приложения, и т.д.),
* какое-то событие происходит на уровне стекла (=жест),
* канал платформы отправляет некоторые данные,
* но также и в основном, когда Engine готов к рендерингу нового кадра.

Таким образом, можно не отказываться от тонны полезного кода, который целое десятилетие создается в сообществе мобильных разработчиков? Все библиотеки, доступные в нативных приложениях SDK и платформенные API могут быть использованы для Flutter-приложений.

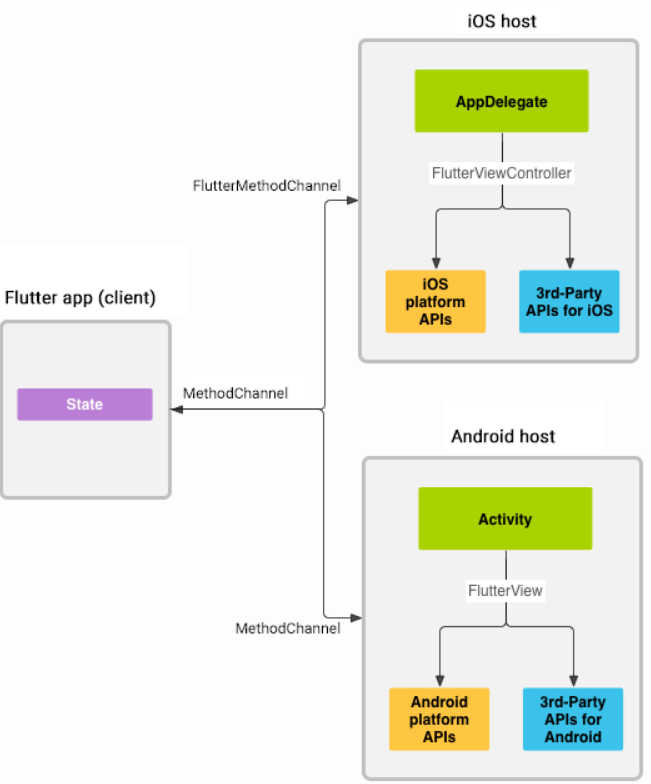


Рисунок 2 – Разделение на IOS host и Android host

Чтобы понять, как работают внутренние механизмы Flutter, рассмотрим рисунок 3.

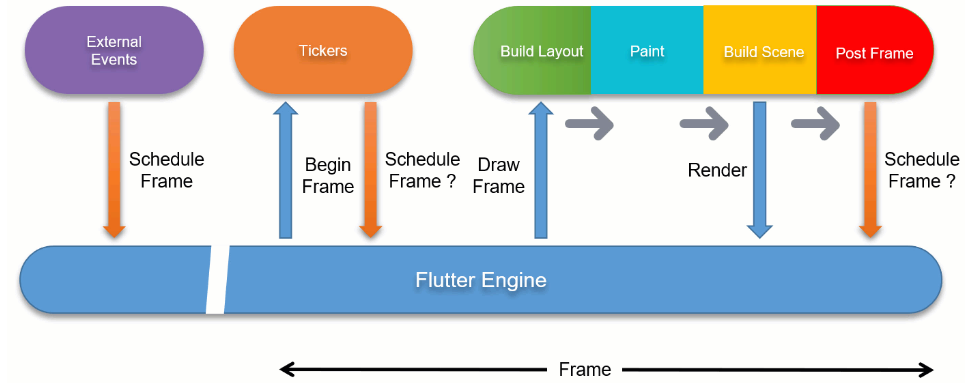


Рисунок 3 – Работа внутренних механизмов Flutter

1. Некоторые внешние события или даже futures могут запускать задачи, которые приводят к необходимости обновления отображения. Соответствующее сообщение отправляется Flutter Engine,
2. Когда Flutter Engine готов приступить к обновлению рендеринга, он создает Begin Frame запрос,
3. Этот Begin Frame запрос перехватывается Flutter Framework, который выполняет задачи, связанные в основном с Tickers,
4. Эти задачи могут повторно создать запрос для более поздней отрисовки,
5. Далее Flutter Engine отправляет Draw Frame, который перехватывается Flutter Framework, который будет искать любые задачи, связанные с обновлением макета с точки зрения структуры и размера,
6. После того, как все эти задачи выполнены, он переходит к задачам, связанным с обновлением макета с точки зрения отрисовки,
7. Если на экране есть что-то, что нужно нарисовать, то новая сцена для визуализации отправляется в Flutter Engine, который обновит экран,
8. Затем Flutter Framework выполняет все задачи, которые будут выполняться после завершения рендеринга (= PostFrame callbacks), и любые другие последующие задачи, не связанные с рендерингом.

После чего процесс выполняется снова.

**Начало разработки Flutter**

При разработке официальная документация советует использовать Android Studio, IntelliJ или VSCode c соответствующими плагинами, но подойдет любой текстовый редактор.

Как приступить к разработке?

***Шаг 1.*** Скачать и распаковать в нужную папку Flutter SDK,

***Шаг 2.*** Для использования Flutter в Android Studio необходимо установить два плагина: Flutter и Dart,

***Шаг 3.*** Создать проект, выполнив в консоли команду flutter create name или используя IDE (в Android Studio → New Flutter Project),

В корневой директории приложения появится 4 пакета — lib, ios, android и test. Первая — директория фреймворка. Там располагаются все dart-файлы и основной код приложения. Несмотря на то, что Flutter компилируется в нативный код, для каждой из платформ приходится писать некоторые нативные взаимодействия. Для нативного кода предусмотрены два пакета –ios/android, в которых есть возможность писать на привычном для конкретной платформы языке — Obj-C/Swift или Java/Kotlin. В пакете test – тесты.

***Шаг 4.*** В проекте сразу будет файл main.dart, содержащий код с примером. У приложения единая точка входа — метод main. Он отвечает за создание корневого виджета. Можно запускать проект.

**Разработка UI**

Любая сущность пользовательского интерфейса – виджет. Текстовое поле, отступ или детектор жестов – виджеты. Приложение построено из них, как конструктор. Согласно официальному сайту, “каждый виджет является неизменяемым описанием части пользовательского интерфейса”.

Рассмотрим приложение, которое изображено на рисунке 4. Данное приложение генерирует случайное имя и выводит его на экран.

На рисунке 4 присутствуют следующие виджеты:

* приложение,
* экран,
* AppBar,
* текст,
* кнопка с лоадером.

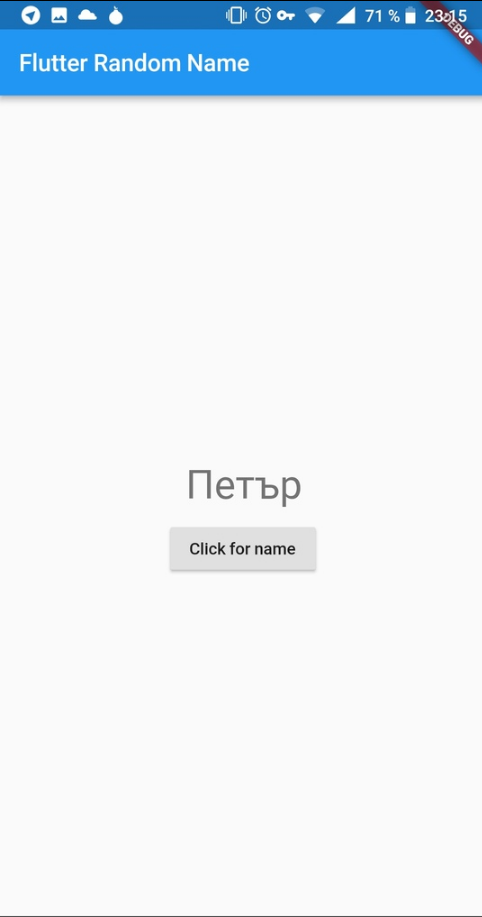


Рисунок 4 – Приложение, генерирующее случайное имя

Часть из них реализована в фреймворке, часть необходимо составить самим из готовых деталей.

Виджеты бывают двух типов ***Stateless*** и ***Stateful***. Первые статичны (например, текст), вторые поддерживают изменение состояния (например, экран).

Чтобы создать ***Stateless*** виджет, нужно:

1. Придумать красивое имя для нового класса;
2. Унаследовать класс от StatelessWidget;
3. Реализовать метод build(), который принимает BuildContext в качестве аргумента и возвращает какой-нибудь Widget.

Пример такого виджета:

class MyStatelessWidget extends StatelessWidget {

final String name;

MyStatelessWidget(this.name);

@override

Widget build(BuildContext context) {

return Text('Hello, $name!');

}

}

**StatefulWidget** — простые, даже проще чем **StatelessWidget**'ы. Но есть нюанс: они не существуют сами по себе, для их работы нужен еще один класс который будет хранить состояние этого виджета. При этом, его визуальная часть (виджеты из которых он состоит) также становятся его состоянием.

Пример такого виджета:

class Counter extends StatefulWidget {

@override

State<Counter> createState() => \_CounterState();

}

Где \_CounterState:

class \_CounterState extends State<Counter> {

int counter = 0;

@override Widget build(BuildContext context) {

return …

}

}

Во Flutter виджеты разделены на 3 категории:

***Proxy:*** Основная задача этих виджетов состоит в том, чтобы хранить некоторую информацию (которая должна быть доступной для виджетов), части древовидной структуры, основанной на Proxy.

***Renderer:*** Данные виджеты имеют непосредственное отношение к компоновке экрана, поскольку они определяют (или используются для определения) размеры, положение, отрисовку.

***Component:*** Это другие виджеты, которые предоставляют непосредственно не окончательную информацию, связанную с размерами, позициями, внешним видом, а скорее данные (или подсказки), которые будут использоваться для получения той самой финальной информации. Эти виджеты обычно называются компонентами.

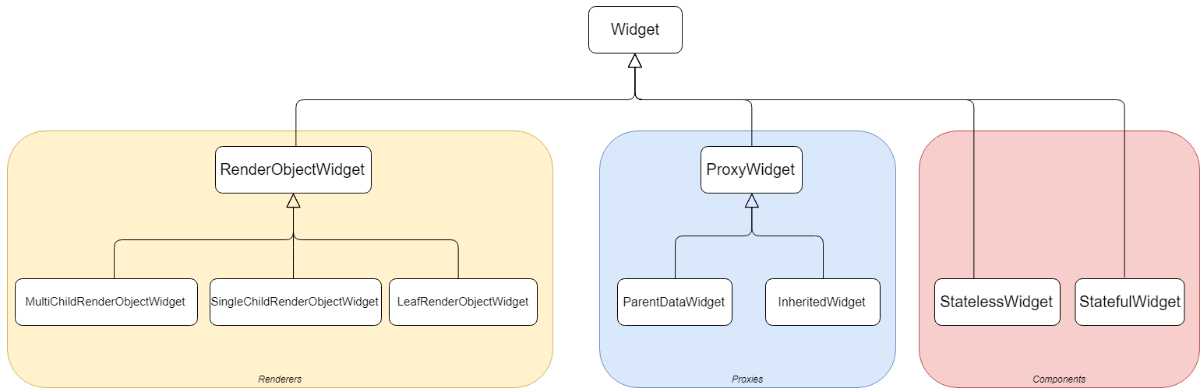


Рисунок 5 – Классификация виджетов

**«За» и «Против» использования Flutter**

***«За»:***

***Хайп***. Технология явно словила волну успеха, имеет активно развивающееся комьюнити и интересные архитектурные подходы в своей основе. Это, на мой взгляд, залог длительного и успешного существования фреймворка.

***Быстрое написание кода***. Собственный рендеринг, независимость от ОЕМ-виджетов и богатый инструментарий разработчика существенно ускоряет написание кода и поддержку двух платформ одновременно.

***Один и тот же код запускается на двух и более платформах.*** На данный момент полноценно поддерживаются только iOS и Android. Web и desktop — в активной разработке. Но их выход в продакшн — вопрос ближайшего будущего.

***Меньше тестирования.*** За счет pixel perfect rendering, тестирование и устранение платформо-зависимых багов и нюансов верстки будет проходить существенно быстрее.

***Быстродействие***. Компиляция в нативный код не оставляет сомнений в этом утверждении.

***Одинаковый UI на всех устройствах.*** При необходимости написать приложение с уникальным дизайном, который должен быть одинаковым на двух мобильных платформах, Flutter становится незаменимым инструментом для кроссплатформенной разработки.

***«Против»:***

***Зрелость***. Молодость платформы — неизбежный спутник Flutter. В этом случае, React Native — более устоявшееся решение, под которое будет легче найти разработчиков.

***Крутая кривая обучения.*** Dart добавляет накладные затраты на внедрение Flutter за счет необходимости изучать и вникать в новый язык.

***Новые нативные UI компоненты.*** В случае с React Native, они могут быть подключены в проект достаточно быстро. Во Flutter все новые появляющиеся на стороне платформы элементы UI должны быть заново отрисованы.

***Необходимость продать технологию менеджеру на текущем этапе ее развития.*** С каждым днем это будет сделать все легче и легче. Но на данный момент — это довольно сильный барьер. Потенциал отправить в Google graveyard. Судя по вложенным ресурсам и PR-активности, это маловероятный исход, но шанс всегда есть.

***Отсутствие вакансий на рынке.*** На данный момент их очень мало. Это должно измениться в будущем. Но сейчас это может стать препятствием для разработчика, который не хочет вкладывать свои силы в изучение новой технологии.

***Опасения, что Flutter не способен закрыть потребности бизнеса.*** Обещая успешную работу на всем, у чего есть GPU и экран, может оказаться запускаемым везде, но не быть востребованным в бизнесе.

**Заключение**

Лично мое мнение заключается в том, что Flatter действительно может очень хорошо себя показать. Скажу, что меня заинтересовал данный SDK и, возможно, в ближайшем будущем я займусь его изучением. Flutter отличается от других технологий и это весьма интересно.

**Источники**

1. [Оффициальный сайт Flatter](https://dart.dev);
2. [Статья на Habr: Как работает Flutter](https://habr.com/ru/post/476018/#chast-1-predystoriya);
3. [Статья на Habr: Flutter — новый взгляд на кроссплатформенную разработку](https://habr.com/ru/company/google/blog/426701/);
4. [Статья на Habr: Про Flutter, кратко: Основы](https://habr.com/ru/post/430918/);
5. [Википедия](https://ru.wikipedia.org/wiki/Flutter_(SDK));
6. [Статья на DOU: Стоит ли инвестировать во Flutter. Сравнение Flutter и React Native](https://dou.ua/lenta/articles/flutter-for-mobile-apps/);