УДК 004.428.4

РАЗРАБОТКА БИБЛИОТЕКИ ДЛЯ ОТОБРАЖЕНИЯ ВЕКТОРНЫХ КАРТ НА FLUTTER

Чуриков Е.А., Зудилова Т.В., Ананченко И.В., Иванов С.Е.
ФГАОУ ВО «Национальный исследовательский университет ИТМО», Санкт-Петербург, e-mail: churikov.egor@bk.ru, zudilova@ifmo.spb.ru, anantchenko@yandex.ru, serg_ie@mail.ru

Рассматривается процесс разработки библиотеки для кроссплатформенного фреймворка Flutter, которая позволяет отображать интерактивные векторные карты и объекты на них. Приводятся причины необходимости векторных карт в отдельных случаях, выявляются ключевые и вторичные требования к планируемой библиотеке, проводится сравнительный анализ существующих на данный момент решений (как пакетов – библиотек, написанных на чистом Flutter и Dart, так и плагинов – библиотек, обращающихся к нативному коду) по реализации библиотек векторных карт на фреймворке Flutter на основе выявленных требований. Приведено описание реализации собственной библиотеки на базе одного из существующих на рынке решений с открытым исходным кодом. Демонстрируемые примеры реализации включают в себя набор методов класса контроллера, необходимых для дальнейшей работы с библиотекой, представлена реализация полностью кастомизируемых маркеров как на основе Flutter-виджетов, так и посредством передачи растрового изображения из файлов проекта или по URL. Созданная библиотека для отображения интерактивных векторных карт может использоваться в различных Flutter-проектах, предполагающих наличие интерактивных карт и создающихся для платформ iOS, Android и Web. Рассматриваемое решение в настоящее время используется в мобильном приложении «Цифровое Приморье», найти и установить которое можно в App Store и Google Play. Обзорно рассматривается процесс анимированного перемещения маркера по карте между двумя географическими координатами формата LatLng. Представлены планы развития и модификации рассматриваемой библиотеки и варианты ее практического применения на реальных Flutter-проектах для операционных систем iOS и Android.

Ключевые слова: векторные карты, библиотека, flutter, mobile, android, ios, mapbox, maplibre

DEVELOPMENT OF A LIBRARY FOR DISPLAYING VECTOR MAPS ON FLUTTER

Churikov E.A., Zudilova T.V., Ananchenko I.V., Ivanov S.E.
ITMO National Research University, Saint-Petersburg, e-mail: churikov.egor@bk.ru, zudilova@ifmo.spb.ru, anantchenko@yandex.ru, serg_ie@mail.ru

The process of developing a library for the cross-platform framework Flutter, which allows you to display interactive vector maps and objects on them, is considered. The reasons for the need for vector maps in some cases are given, key and secondary requirements for the planned library are identified, a comparative analysis of currently existing solutions (both library packages written in pure Flutter and Dart, and plug–ins libraries accessing native code) for the implementation of vector map libraries on the Flutter framework based on identified requirements. The article describes the implementation of its own library based on one of the open-source solutions available on the market. The demonstrated implementation examples include a set of controller class methods necessary for further work with the library, the implementation of fully customized markers is presented both on the basis of Flutter widgets and by transferring a bitmap image from project files or by URL. The created library for displaying interactive vector maps can be used in various Flutter projects involving interactive maps and created for iOS, Android and Web platforms. The solution in question is currently being used in the “Digital Primorye” mobile application, which can be found and installed in the App Store and Google Play. The process of animated marker movement on the map between two geographical coordinates of the LatLng format is reviewed. Plans for the development and modification of the library under consideration and options for its practical application on real Flutter projects for iOS and Android operating systems are presented.

Keywords: vector maps, library, flutter, mobile, android, ios, mapbox, maplibre

Интерактивные карты занимают особое место в современных мобильных приложениях. Приложения могут строиться вокруг интерактивной карты как главного модуля и основного инструмента взаимодействия пользователя с приложением. Существует множество вариаций интерактивных карт, но в данной статье рассматриваются именно географические карты, отображающие ландшафт местности и объекты на ней. На данный момент существует 2 самых популярных типа географических карт: растровые тайлы и векторные карты. Растровые тайлы – растровые квадратные изображения, из которых на основе сетки строится изображение карты, которое видит пользователь на экране смартфона. Векторные карты – более современное и интеллектуальное решение для схематического отображения местности и объектов на ней, при котором с сервера заранее загружается конфигурационный файл формата json, содержащий конфигурацию стилей карты – набор правил, согласно которым будут отображаться различные объекты. По мере взаимодействия пользователя с векторной

картой, с сервера приходят данные об объектах, видимых пользователю, сразу после чего к этим данным применяются полученные ранее стили и передаются на движок рендеринга, задачей которого является отображение данных в привычном для пользователя виде. Если проводить сравнение данных подходов, векторные карты не отображают информацию в виде растровых изображений, что может увеличить скорость отображения объектов на карте, однако будет более тяжеловесным в плане производительности, поскольку есть необходимость рендеринга в реальном времени. Далеко не каждая команда разработки может позволить себе собственноручно реализовать векторные карты из-за высокого уровня сложности реализации, поэтому нередко команды прибегают к использованию существующих решений.

Цель выполненной работы – продемонстрировать новый подход к реализации собственной библиотеки, дающей возможность добавить во Flutter-проект гибко настраиваемую векторную карту с высокой степенью интерактивности объектов на них.

История

Работая над мобильным приложением, которое было написано на кроссплатформенном фреймворке Flutter [1] под операционные системы iOS и Android и должно было содержать интерактивную карту, столкнулись с проблемой медленной загрузки тайлов карты через Интернет. Проблема проявлялась следующим образом:

1. Пользователь открывает карту, видит белый экран-подложку и не видит никакой карты.

2. Спустя пару секунд по мере загрузки начинают по очереди появляться квадратные области с картой. Такую схему визуализации едва ли можно назвать удачной.

3. Если резко приблизить уже загруженный тайл, он увеличивается в размере (как обычное растровое изображение), в это время отправляется запрос к тайловому серверу (тайловый сервер – удаленная логическая единица, принимающая в качестве параметра координаты двух географических точек (например, правая верхняя и левая нижняя, или наоборот) и возвращающая набор тайлов) на получение новых тайлов по новым крайним координатам. До этого времени пользователь видит растянутую на большую часть экрана картинку (тайл) в довольно низком разрешении, что также негативно сказывается на степени интерактивности карты и ухудшает восприятие изображения пользователем. Описываемую проблему с изображением видно на рисунке 1.

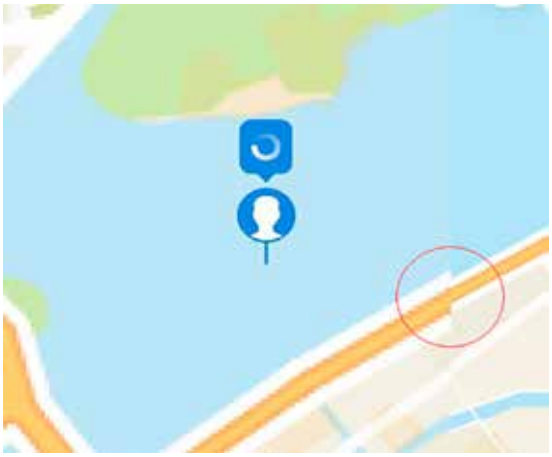

Рис. 1. Проблемы при медленной загрузке растровых тайлов

Проблема на стороне тайлового сервера, поэтому её решение невозможно со стороны мобильного клиента. Поскольку как-либо оптимизировать тайловый сервер не представлялось возможным, пришлось прибегнуть к альтернативному варианту, а именно к векторным картам. Главная идея заключалась в том, что не придется передавать с клиента на сервер мегабайты растровых тайлов (по сути, изображений в формате PNG) на низкой скорости. Вместо этого с сервера на клиент приходила лишь описательная информация в формате JSON о том, какие объекты (леса, горы, реки, дороги, здания и т.д.) находятся на конкретных координатах, а рендер изображения по полученным данным является уже обязанностью мобильного клиента. К тому же на сервере уже была реализована передача данных и стилей на клиент в формате JSON.

Поиски готового решения

Для начала было проведено исследование существующих на данный момент библиотек [2] для отображения векторных карт на Flutter. После его окончания было выделено несколько главных претендентов: google_maps_flutter, flutter_map (вместе с пакетом flutter_vector_map_tiles), mapbox_gl, flutter-maplibre-gl, syncfusion_flutter_maps и yandex_mapkit. Вышеперечисленные библиотеки написаны для использования их с Flutter и Dart. Помимо главных требований совместимости с фреймворком и отображения векторных карт, были и вторичные, но, как оказалось, не менее важные требования к проприетарности библиотеки. Условие было простым: необходима библиотека с открытым исходным кодом, бесплатная для коммерческого использования.

Сравнение плагина flutter-maplibre-gl и пакета flutter_vector_map_tiles

	flutter-maplibre-gl	flutter_vector_map_tiles
Создание группы маркеров	+	+
Анимированное перемещение маркера между координатами	Нужно реализовывать самостоятельно	Нужно реализовывать самостоятельно
Кастомизация маркеров	Маркеры могут быть либо ассетами, либо ссылкой на url картинки в Интернете, либо растризованным набором битов	Основан на flutter_map, поэтому маркеры это виджеты, что дает широкие возможности для кастомизации маркеров
Добавление полигонов	+	+
Добавление полилайнов	+	+

По одной или нескольким из вышеперечисленных причин флагманы данной подборки google maps, mapbox, yandex и syncfusion были отклонены. Для оставшихся библиотек было проведено сравнительное тестирование с нагрузкой, подобной рабочей. Были созданы демонстрационные мобильные приложения, показывающие возможности каждой из двух библиотек flutter-maplibre-gl и flutter_map (вместе с пакетом flutter_vector_map_tiles) применительно к рабочим задачам. Например, создание группы маркеров, их анимированное перемещение между координатами, возможности при кастомизации внешнего вида и поведения маркеров, добавление на карту полигонов и полилайнов. Авторами составлена сравнительная таблица возможностей двух выше представленных библиотек для интеграции в приложение векторных карт flutter-maplibre-gl и flutter_map (вместе с пакетом flutter_vector_map_tiles).

На следующем этапе был выполнен тест производительности, в рамках которого в каждом из двух демонстрационных приложений реализована возможность добавлять на карту большое количество объектов (маркеров, полигонов, полилайнов), чтобы посмотреть, как будет меняться производительность приложения в зависимости от количества добавленных на карту точек. На этом этапе выяснилось, что пакет flutter_vector_map_tiles, воспроизводящий векторные карты при помощи графического движка рендеринга Skia [3], используемого для фреймворков Flutter, при запуске на Android не может обеспечить должный уровень производительности на уровне хотя бы 30 кадров/с даже без добавленных на карту маркеров. Следовательно, дальнейшее его использование в проектах нецелесообразно, поскольку в рабочей нагрузке при добавлении объектов на карту производительность будет только ухудшаться. Выбор был сделан в пользу flutter-maplibre-gl, который выдавал относительно стабильный фремрейт в районе 60 кадров/с.

Плагин flutter-maplibre-gl – библиотека для Flutter с открытым исходным кодом, построенная на базе плагина Mapbox GL, являющегося плагином для Flutter, который реализует взаимодействие с нативными плагинами Mapbox GL Native для операционных систем iOS и Android, а также с библиотекой Mapbox GL JS для Web. Его главная особенность – отсутствие проприетарности компании Mapbox, что выражается в возможности использовать данную библиотеку без токена Mapbox API Key. К сожалению, при более детальном рассмотрении выявились некоторые особенности, которые в теории могут замедлять процесс создания мобильных приложений на его основе: сложное API, отсутствие полностью кастомизируемых маркеров на основе виджетов Flutter, а также отсутствие удобного механизма анимированного перемещения маркеров между двумя координатами. Недостатки было решено исправить путём создания собственного пакета [4] для Flutter на основе плагина flutter-maplibre-gl, который бы исправлял отмеченные недочеты.

Создание собственного решения на базе плагина flutter-maplibre-gl

Сначала был создан Flutter-виджет FwdMap, который представляет собой StatefulWidget, принимает необходимые параметры (которые позже будут переданы в виджет MaplibreMap), содержит минимальную логику для обновления состояния карты и в методе build реализует Stack виджетов, среди которых виджет векторной карты MaplibreMap и слои динамических и статических маркеров. Ниже представлен фрагмента кода метода build виджета FwdMap.

```
@override
 Widget build(BuildContext context) {
  return Stack(
   children: [
    MaplibreMap(
     styleString: widget.styleString,
     trackCameraPosition: true,
     onMapCreated: _onMapCreated,
     onMapLongClick: widget.onMapLongClick,
     onCameraIdle: widget.onCameraIdle,
     onStyleLoadedCallback: widget.onStyleLoadedCallback,
     initialCameraPosition: widget.initialCameraPosition,
     minMaxZoomPreference: MinMaxZoomPreference(widget.minZoom, widget.maxZoom),
    ),
    if (_dynamicMarkerAnimationWidgets.isNotEmpty) ..._dynamicMarkerAnimationWidgets.values.toList(),
    if (_staticMarkerAnimationWidgets.isNotEmpty) ..._staticMarkerAnimationWidgets.values.toList(),  ], );  }
```

Затем необходимо было реализовать собственную версию контроллера карты, которая бы, в отличие от MaplibreMapController, предоставляла удобное API для главных функций – добавления, изменения и удаления объектов, а также их анимированного перемещения между координатами и анимированного изменения позиции камеры на карте. Для каждого типа объектов было создано по контроллеру, чтобы разделить обязанности согласно принципу разделения интерфейсов из SOLID [5]. Следом были реализованы главные методы FwdMapController (некоторые из них вызывают методы вложенных контроллеров), которые реализуют основной требуемый функционал карты:

• addStaticMarker (+ метод updateStaticMarker) – добавляет на карту так называемый статический маркер (т.е. маркер, нативно добавляемый на карту средствами плагина flutter-maplibre-gl);

• addDynamicMarker (+ метод updateDynamicMarker) – добавляет на карту динамический маркер (т.е. маркер, добавляемый с помощью Stack поверх виджета карты средствами фреймворка Flutter);

• addPolyline (+ метод updatePolyline) – добавляет на карту полилайн средствами плагина flutter-maplibre-gl;

• addPolygon (+ метод updatePolygon) – добавляет на карту полигон средствами плагина flutter-maplibre-gl;

• remove – удаляет объект с карты по id;

• clearMap – удаляет все объекты с карты;

• animateMarker – анимированно перемещает маркер между координатами;

• getUserLocation – получает текущую геопозицию пользователя.

Рассмотрим подробнее классы маркеров, полигона и полилайна. Первым рассматривается класс статического маркера, который в числе прочего принимает на вход координаты и объект класса FwdStaticMarkerImage, который в свою очередь определяет способ создания статического маркера с помощью реализованной утилиты FwdMapMarkerHelper (image asset, image url, растризованный виджет Flutter). Ниже представлен код класса FwdStaticMarkerImage.

```
class FwdStaticMarkerImage {
 final Uint8List _bytes;
 final String name;
 Uint8List get bytes => _bytes;
 FwdStaticMarkerImage._({    required Uint8List bytes,    required this.name,  }) : _bytes = bytes;
 static Future<FwdStaticMarkerImage> fromWidget({
  required Widget child,    String? cacheKey,    Duration? creatingImageDelay,
  MarkerAnchor anchor = MarkerAnchor.center,  }) async {
  final widgetBytes = await FwdMapMarkerHelper.widgetToBytes(child, delay: creatingImageDelay,
cacheKey: cacheKey);
  return FwdStaticMarkerImage._(
   bytes: widgetBytes,
   name: cacheKey ?? FwdId.generateFromRandomUUID().toString(),    );  }
 static Future<FwdStaticMarkerImage> fromImageAsset({
  required String imageAssetPath,
  required String cacheKey,  }) async {
  final imageBytes = await FwdMapMarkerHelper.imageAssetToBytes(imageAssetPath, cacheKey: cacheKey);
  return FwdStaticMarkerImage._(bytes: imageBytes, name: cacheKey);  }
 static Future<FwdStaticMarkerImage> fromImageNetwork({    required String imageUrl,  }) async {
  final imageBytes = await FwdMapMarkerHelper.imageNetworkToBytes(imageUrl);
  return FwdStaticMarkerImage._(bytes: imageBytes, name: imageUrl);  } }
```

Класс утилиты FwdMapMarkerHelper реализует 3 статичных метода для получения массива байтов изображения из различных источников, таких как ассет (предзагруженная в проект картинка), изображение в Интернете или растризованный виджет Flutter. Растризация Flutter виджета реализована с помощью пакета screenshot, один из методов которого принимает на вход Flutter виджет, а возвращает уже его растризованную версию, т.е. массив байтов. Изображение из сети Интернет получается с помощью пакета http, а из ассета массив байтов получается стандартными средствами Flutter и Dart (rootBundle). Ниже представлен код класса утилиты FwdMapMarkerHelper.

```
class FwdMapMarkerHelper {
 static Map<String, Uint8List> cachedWidgets = {};
 static Future<Uint8List> widgetToBytes(Widget widget, {Duration? delay, String? cacheKey}) async {
  if (cachedWidgets.containsKey(cacheKey)) {  return cachedWidgets[cacheKey]!;  }
  final bytes = await ScreenshotController().captureFromWidget(
   widget,  delay: delay ?? const Duration(seconds: 1),  );
  if (cacheKey != null) {  cachedWidgets[cacheKey] = bytes;  }  return bytes;  }
 static Future<Uint8List> imageAssetToBytes(String imageAssetPath, {String? cacheKey}) async {
  if (cachedWidgets.containsKey(cacheKey)) {  return cachedWidgets[cacheKey]!;  }
  final Uint8List bytes = (await rootBundle.load(imageAssetPath)).buffer.asUint8List();
  if (cacheKey != null) {  cachedWidgets[cacheKey] = bytes;  }  return bytes;  }
 static Future<Uint8List> imageNetworkToBytes(String imageUrl) async {
  final response = await http.get(Uri.parse(imageUrl));  return response.bodyBytes;  }}
```

Динамический маркер (DynamicMarker) представляет собой Flutter-виджет, отображаемый с помощью встроенного виджета Flutter, Stack, поверх виджета карты с динамически изменяющейся позицией на экране при перемещении камеры карты. Чтобы это стало возможным, пришлось переопределить позиционирование виджета на экране и добавить логику, привязывающую позицию виджета на экране к координатам на карте. Ниже представлен код метода build класса FwdDynamicMarker.

```
@override
Widget build(BuildContext context) {
var ratio = 1.0;
Platform._operatingSystem
  if (!kIsWeb) {
   // iOS returns logical pixel while Android returns screen pixel
   ratio = Platform.isIOS ? 1.0 : MediaQuery.of(context).devicePixelRatio;  }
  return Positioned(  left: _position.x / ratio - 50 / 2,
   top: _position.y / ratio - 50 / 2,  child: Transform.rotate(  angle: -_bearing * pi / 180,
    child: GestureDetector(  onTap: () {
      widget.onMarkerTap?.call(widget.id, widget.coordinate, _position);  },
     child: widget.child,  ),  ),  ); }}
```

Ниже представлен код метода класса FwdDynamicMarker, определяющий позицию на экране (ссылающийся на аналогичный метод плагина flutter-maplibre-gl)

```
 Future<void> calculatePosition() async {
  _position = await widget.maplibreMapController.toScreenLocation(widget.coordinate);  }
```

Статические маркеры добавляются нативно прямо на слой карты, как это делается в flutter-maplibre-gl (но удобнее, поскольку есть только один способ добавить маркеры на карту – через метод контроллера), тогда как динамические маркеры отображаются поверх виджета карты, а изменение их позиции на экране переопределено. Более подробную реализацию можно посмотреть по ссылке на GitHub [4].

Анимация перемещения маркера между начальной и конечной координатами реализована внутри отдельного ответственного за это класса контроллера FwdMarkerAnimationController, реализующего метод animate, который, используя класс Flutter для контроля анимаций AnimationController, изменяет исходные координаты точки в диапазоне от начального до конечного значения с постоянным шагом и заданным временем исполнения, при этом обновляя состояние на каждой итерации для плавного отображения анимации.

Что касается полигонов и полилайнов, они практически не претерпели изменений

по сравнению с плагином flutter-maplibre-gl. Отметим, что появились соответствующие классы FwdPolygon и FwdPolyline, которые являются единицей хранения и транспортировки данных об этих объектах. Добавление (а также изменение и удаление) полигонов и полилайнов на карту происходит посредством MaplibreMapController из оригинального плагина. На рисунке 2 показан результат добавления маркеров, полигонов и полилайнов на интерактивную карту.

Рис. 2. Использование библиотеки в мобильном приложении «Цифровое Приморье»

Фрагмент кода класса FwdPolygon:

```
class FwdPolygon {
final FwdId id;  final List<List<LatLng>> geometry;
final double? borderThickness;  final Color? fillColor;
final Color? borderColor;  final Function(FwdId,
Point<double>, LatLng)? onTap;}
```

Фрагмент кода класса FwdPolyline:

```
class FwdPolyline {
final FwdId id;  final List<LatLng> geometry;
final double? thickness;  final Color? color;
final Function(FwdId, Point<double>, LatLng)?
onTap;}
```

Выводы

Выполнен сравнительный анализ существующих реализаций пакетов и плагинов векторных карт для фреймворка Flutter, предложена собственная интерпретация существующего подхода для достижения таких результатов, как кастомизация маркеров, анимированное перемещение маркеров, повышение удобства разработки путем упрощения API. Создана библиотека для отображения интерактивных векторных карт, которая может использоваться в самых различных Flutter-проектах, предполагающих наличие интерактивных карт и создающихся для платформ iOS, Android и Web. На момент написания статьи рассмотренное решение используется в мобильном приложении «Цифровое Приморье», найти и установить которое можно в App Store [6] и Google Play [7].

В качестве логичного продолжения работы планируется создание полноценного демонстрационного Flutter-проекта для платформ iOS, Android и Web с последующим добавлением его в GitHub-репозиторий. Также в планах обновлять получившуюся библиотеку в соответствии с выходящими обновлениями плагина flutter-maplibre-gl, на основе которого построено представленное решение. Созданная библиотека доступна для скачивания и клонирования по ссылке на GitHub и может быть использована, в том числе в коммерческих целях, принимаются предложения по улучшению и оптимизации в виде Pull Request.

Список литературы

1. Чуриков Е.А., Зудилова Т.В., Ананченко И.В., Осипов Н.А., Иванов С.Е., Осетрова И.С. Сравнение современных средств разработки мобильных приложений // Современные наукоемкие технологии. 2022. № 12-1. С. 82-87.

2. Top Flutter Geolocation and Maps packages // Flutter Gems. URL: https://fluttergems.dev/geolocation-maps/ (дата обращения: 22.02.2023).

3. Biessek A. Flutter for Beginners: An introductory guide to building cross-platform mobile applications with Flutter 2.5 and Dart. 2-е изд. г. Бирмингем, Объединенное Королевство Великобритании: Packt Publishing, 2021. 370 с.

4. fwd_map // GitHub. URL: https://github.com/Churikov0112/fwd_map (дата обращения: 20.02.2023).

5. Robert C. Martin Clean Code: A Handbook of Agile Software Craftsmanship. London: Pearson, 2008. 464 с.

6. Цифровое Приморье // App Store. URL: https://apps.apple.com/ru/app/цифровое-приморье/id1640251581 (дата обращения: 20.02.2023).

7. Цифровое Приморье // Google Play. URL: https://play.google.com/store/apps/details?id=team.nineone.cp_app.prod&hl=ru&gl=US (дата обращения: 20.02.2023).