<https://zen.yandex.ru/media/id/5bbcbc1ba5bd5400a990e7d9/pogrujenie-v-jetpack-compose-5d22ecd1ecbe4e00ad59c180>

**Погружение в Jetpack Compose**

8 июля 2019

161 дочитывание

2,5 мин.

262 просмотра. Уникальные посетители страницы.

161 дочитывание, 61%. Пользователи, дочитавшие до конца.

2,5 мин. Среднее время дочитывания публикации.



***Перевод статьи******подготовлен******для студентов******продвинутого курса***[***«****Android-разработчик****»***](https://otus.pw/MNKo/)***в образовательном проекте OTUS.***

**Пробуем новый UI-фреймворк для Android-приложений**

В течение последних нескольких лет, участвуя во многих мобильных проектах, мне приходилось использовать различные технологии, например, такие как Android, ReactNative и Flutter. Переключение с ReactNative обратно на классический Android вызвало у меня смешанные чувства. Возвращение к Kotlin прошло хорошо, но я очень скучал по UI-фреймворку React. Небольшие повторно используемые компоненты, с помощью которых создается пользовательский интерфейс, великолепны и обеспечивают большую гибкость и скорость разработки.

Вернувшись в классический Android, мне нужно было беспокоится о том, чтобы сохранить иерархию View как можно более однообразной. Из-за этого трудно по-настоящему посвятить себя компонентному подходу. Это делает копипаст более заманчивым, что приводит к более сложному и менее поддерживаемому коду. В конечном итоге мы удерживаем себя от экспериментов с пользовательским интерфейсом, которые могли бы улучшить UX.



Android раскрывает Jetpack Compose. Иллюстрация: Эмануэль Багилла (Emanuel Bagilla)

**Jetpack Compose спешит на помощь**

Поэтому после просмотра [What’s new in Android](https://www.youtube.com/watch?v=td3Kd7fOROw" \t "_blank) с конференции Google I/O 2019 я сразу же начал разбираться с Compose и постарался больше узнать о нем. Compose — это инструментарий реактивного пользовательского интерфейса, полностью разработанный на Kotlin. Compose выглядит очень похоже на существующие фреймворки пользовательских интерфейсов, такие как React, Litho или Flutter.

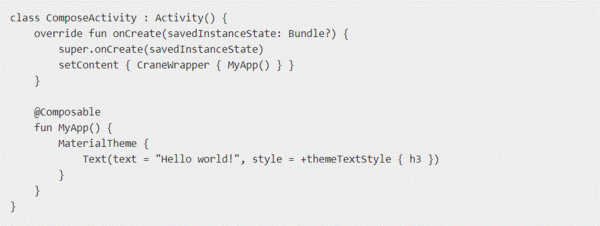
Нынешняя структура UI-фреймворка Android существует с 2008 года, и со временем стала более сложной, ее довольно тяжело поддерживать. Jetpack Compose стремится начать все с начала с учетом философии современных компонентов. Фреймворк написан с учетом следующих основных целей:

* **Несвязанность с релизами платформы:** Это позволяет быстро исправлять ошибки, поскольку Compose не зависит от новых релизов Android.
* **Меньший стек технологий:** Фреймворк не заставляет вас использовать View или Fragment при создании пользовательского интерфейса. Все элементы являются компонентами и свободно могут быть составлены вместе.
* **Прозрачные управление состояниями и обработка событий:** Одна из самых важных и сложных вещей, которую необходимо решать в больших приложениях, — это обработка потока данных и состояния в вашем пользовательском интерфейсе. Compose проясняет кто отвечает за состояние и как должны обрабатываться события, подобно тому, как это обрабатывает React.
* **Написание меньшего количества кода:** Написание пользовательского интерфейса в Android обычно требует большого количества кода, особенно при создании более сложных layout’ов, например, с помощью RecyclerView. Compose призван значительно упростить способ создания пользовательского интерфейса.

Это облегчает создание изолированных и повторно используемых компонентов, что упрощает создание нового экрана с существующими элементами. Помогая вам, как разработчику, сосредоточиться на создании удобного пользовательского интерфейса, вместо того чтобы пытаться контролировать View-иерархию и приручать View и Fragment.

**Простое приложение с Compose: Hello World**

Давайте посмотрим на код простого приложения «Hello World» с Jetpack Compose.

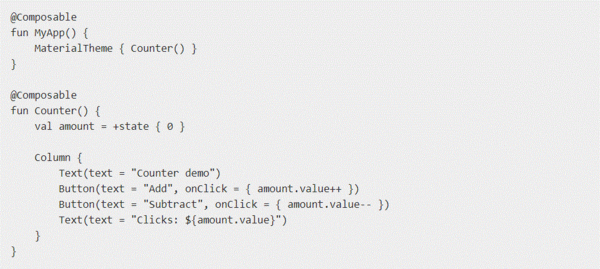


В методе onCreate мы задаем содержимое нашего приложения, вызывая setContent. Это метод, который инициализирует составное дерево виджетов и оборачивает его в FrameLayout.

Чтобы все заработало, нам нужно обернуть наше приложение в CraneWrapper и MaterialTheme. CraneWrapper отвечает за настройку поставщиков для Context, FocusManager и TextInputService. MaterialTheme необходим для предоставления цветов, стилей и шрифтов ваших виджетов. Имея это в виду, мы можем добавить компонент Text, который будет отображать наш текст на экране в определенном стиле.

**Введение состояния**

Управление потоком данных и состояниями может быть сложной задачей. Чтобы проиллюстрировать, насколько это легко с Compose, давайте создадим простое приложение-счетчик.  
Для работы с состояниями Jetpack Compose использует идеи других современных UI-фреймворков, таких как Flutter и React. Существует однонаправленный и реактивный поток данных, который заставляет ваш виджет обновляться или «перекомпоновываться».



В примере выше мы добавляем кнопки «Add» и «Subtract» вместе с лейблом, отображающим текущее количество нажатий. Как вы можете видеть в примере ниже, обновляя состояние «amount», виджеты разумно перекомпоновываются при изменении состояния.



Запуск демо-приложения

Состояние amount инициализируется с помощью +state { 0 }. Пытаясь выяснить, что это за колдовство, я залез в исходный код. Это мое мнение, хотя я все еще не уверен, что до конца все понимаю.

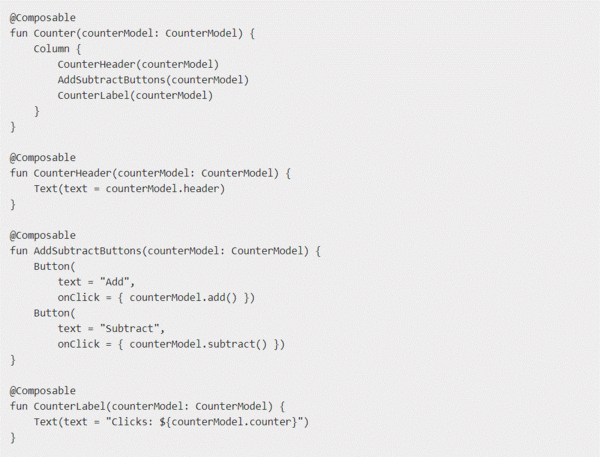
state {...} создает Effect<State<T<code>>. Класс Effect является нечетким классом, который содержит блок исполняемого кода, который выполняется позиционно в контексте композиции. Класс State содержит одно значение с типом Model, по сути делая это значение наблюдаемым (observable). Оператор + является временным оператором, который разрешает State из Effect.

**Пользовательские модели состояний**

Вместо использования +state {} для создания модели отдельного значения мы также можем создать пользовательскую модель с помощью аннотации @Model. Мы можем улучшить наше приложение-счетчик, разделив его на более мелкие виджеты и передав модель другим виджетам, которые обновляются и отображают состояние этой модели.



Используя аннотацию @Model, плагин Compose Compiler делает все переменные в вашей модели наблюдаемыми, чтобы их можно было использовать для перекомпоновки виджетов. Давайте обновим наш виджет, чтобы использовать CounterModel:

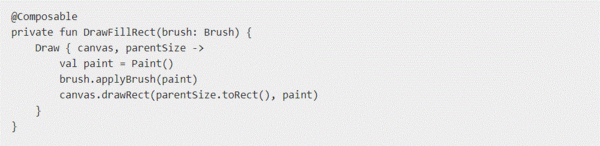


Единственный виджет, из которого состоит приложение-счетчик, теперь разделен на несколько меньших компонуемых виджетов. CounterModel передается различным виджетам, либо для отображения состояния модели, либо для изменения состояния модели с помощью функций add() или subtract().

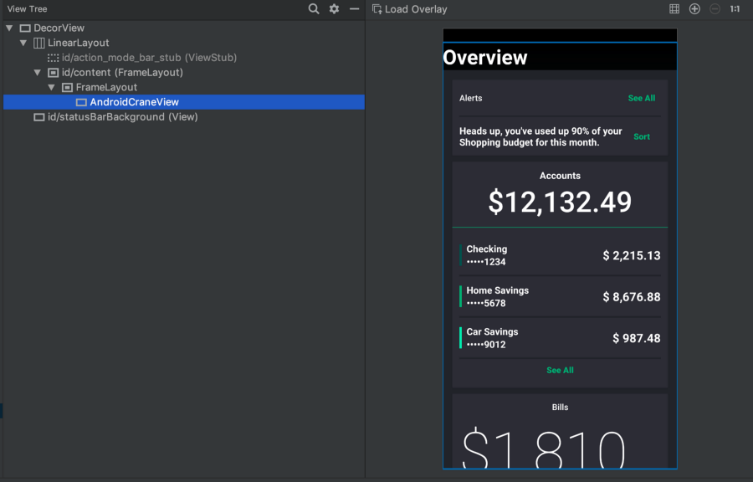
**Больше никаких view**

Важно понимать, что виджеты Jetpack Compose не используют под капотом view или fragment, это всего лишь функции, которые рисуют на холсте. Плагин Compose Compiler обрабатывает все функции с аннотацией @Composable и автоматически обновляет UI-иерархию.

Например, виджет Divider состоит из виджета Padding, который содержит виджет DrawFillRect. Глядя на исходный код DrawFillRect, становится ясно, что он рисует линии прямо на холсте. Все остальные виджеты реализованы таким же образом.



Исходный код DrawFillRect, который используется внутри виджета Divider  
Если мы посмотрим на Layout Inspector, запустив один из примеров приложений от Google, ясно увидим, что при запуске приложения Android с Compose нет никаких View или ViewGroups. Мы видим FrameLayout, содержащий CraneWrapper, который мы создали в коде, оттуда на экране отображается иерархия Compose UI.

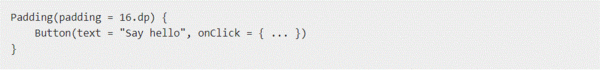


Layout Inspector инспектирует приложение Jetpack Compose.

Отсутствие views также означает, что Jetpack Compose не может использовать доступные в настоящее время view, такие как android.widget.Button, и должен создавать все виджеты с нуля. Если взглянуть, например, на Flutter, в котором используется тот же подход, то можно увидеть, что это тяжелая работа. Это одна из причин почему Jetpack Compose понадобится время, прежде чем он будет готов к использованию в продакшене.

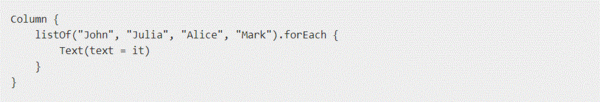
**Все элементы являются виджетами**

Совсем как у Flutter, в Compose все элементы — это виджеты. Более сложные виджеты были разбиты на элементарные виджеты с четкими обязанностями. Поэтому даже padding, spacers и так далее являются виджетами. Например, если вы хотите добавить отступ вокруг кнопки, просто оберните ее в padding виджет:



**Соединение кода с пользовательским интерфейсом**

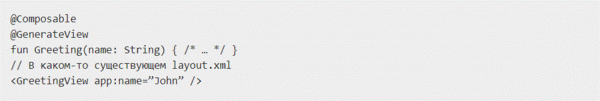
Соединять Kotlin-код с UI-виджетами очень легко. Например, если вы хотите показать пользовательский интерфейс, которые повторяется или зависит от каких-то условий. Так, вы можете легко отобразить список имен, как показано ниже.



Это действительно мощная фича, но вы должны быть осторожны, чтобы не запрограммировать слишком много логики на уровне пользовательского интерфейса.

**Совместимость с вашими Android-приложениями**

Compose разработан таким образом, что вы можете добавить его в уже существующее приложение и постепенно перенести некоторые части вашего UI в новый фреймворк. Приведенные выше примеры добавляют Jetpack Compose UI к одному activity. Также вы можете встроить Compose-виджеты в существующий XML layout с помощью аннотации GenerateView:



**Заключение**

Я в восторге от Compose, потому что он уменьшает растущие страдания, которую я испытываю при разработке под Android. Он помогает быть более гибким, фокусироваться на создании удобного пользовательского интерфейса, а четкая ответственность также позволяет избежать ошибок.

У Compose впереди долгий путь, на мой взгляд, его можно будет использовать в продакшене не раньше чем через год или два. Тем не менее, я думаю, что сейчас подходящий момент, чтобы взглянуть на Jetpack Compose. Создатели активно ищут фидбек, на этом этапе все еще можно вносить изменения. Все отзывы помогут улучшить этот новый фреймворк.

Прочитайте мою статью [«Try Jetpack Compose today»](https://medium.com/q42-engineering/try-jetpack-compose-today-a12bda50aed2), чтобы узнать, как подключить пре-альфа Compose. Также, я думаю, вам очень интересно будет посмотреть [видео по шаблонам декларативного интерфейса](https://youtu.be/VsStyq4Lzxo) с Google I/O.  
Я с нетерпением жду, когда смогу использовать Compose в реальных Android-приложениях!