Введение в виджеты
- Flutter

- Виджеты — это строительные блоки, которые, разработчик, собирает вместе, чтобы создать пользовательский интерфейс. Во Flutter, довольно часто используют фразу «все — это виджет». Каждый объект в пользовательском интерфейсе приложения Flutter является виджетом. Структура определяется виджетами, стили определяются виджетами, даже анимация и маршрутизация обрабатываются виджетами. А виджеты — это всего лишь классы Dart, которые знают, как описать свое представление.

Встроенные виджеты

Flutter предлагает несколько готовых виджетов, которые можно разделить по принципу их использования.

- **Макет** - Row, Column, Scaffold,Stack
- **Структуры** - Button, Toast,MenuDrawer
- **Стили** - TextStyle,Color
- **Анимации** - FadeInPhoto, transformations
- **Позиционирование и выравнивание** - Center, Padding

Написание виджета в коде

- Большая часть кода Flutter, собственные виджеты. Виджеты — это просто класс Dart, расширяющий класс Widget.
- При создании виджетов есть одно требование, налагаемое суперклассом виджетов. У него должен быть build метод. Этот метод должен возвращать другие виджеты. Это метод, используется для объединения виджетов в пользовательский интерфейс.

```
class AlertButton extends StatelessWidget {
  const AlertButton({super.key});

  @override
  Widget build(BuildContext context) => throw UnimplementedError();
}
```

Типы виджетов: с сохранением состояния (StatefulWidget) и без сохранения состояния (StatefulWidget)

- Виджеты Flutter должны расширять несколько классов из библиотеки Flutter. Два которые используются чаще всего, — это StatelessWidget и StatefulWidget.
- State разница в том, что внутри виджета есть концепция , которую можно использовать, чтобы сообщить Flutter, когда выполнять рендеринг и повторный рендеринг.

StatelessWidget

- StatelessWidget— это виджет, в котором не хранится никакой информации, которая в случае потери будет иметь значение. В него передается все состояние или конфигурация виджета. Его единственная задача — отображать информацию и пользовательский интерфейс. Он не сообщает платформе, когда удалить его из дерева или когда перестроить. Скорее, структура сообщает ему, когда следует перестроить.
- Для написания StatelessWidget требуется расширить правильный класс и включить build метод.

```
class TitleText extends StatelessWidget {
  final String text;

  const TitleText(this.text, {super.key});

  @override
  Widget build(BuildContext context) {
    return Center(
      child: Text(text)
    ); // Center
  }
}
```

StatefulWidget

StatefulWidget немного другое. На самом деле это два класса: State объект и сам виджет. Цель этого класса — сохранить состояние, когда Flutter перестраивает виджеты.

Во Flutter концепция состояния определяется двумя вещами:

- Данные, используемые виджетом, могут измениться.
- Данные не могут быть прочитаны синхронно при построении виджета. (Все состояния должны быть установлены к моменту build вызова метода).

Объект State уникален тем, что у него есть несколько методов, которые по-разному взаимодействуют с Flutter. Самый важный из них setState.

setState используется, чтобы сообщить Flutter, что ему необходимо перестроить, обычно потому, что что-то изменилось, и экран должен это отразить. После setState вызова Flutter знает, что ему нужно build снова вызвать метод.

Важно!

- Класс StatefulWidget должен реализовать createState метод.
- Объект State должен реализовывать build метод.
- Неизменно StatefulWidget.
- Объект State изменчив.

Жизненный цикл StatefulWidget

- Когда Flutter создает объект StatefulWidget, он создает State объект. В этом объекте хранятся все изменяемые состояния этого виджета.
- Понятие state определяется двумя вещами:
- Данные, используемые виджетом, могут измениться.
- Данные не могут быть прочитаны синхронно при построении виджета. (Все состояния должны быть установлены к моменту build вызова метода).

Жизненный цикл состоит из следующих упрощенных этапов:

- createState()
- mounted == true
- initState()
- didChangeDependencies()
- build()
- didUpdateWidget()
- setState()
- deactivate()
- dispose()
- mounted == false

Почему StatefulWidget и State — это отдельные классы?

- Одним словом: производительность.
- Версия заключается в том, что State объекты долговечны, но StatefulWidget объекты (и все Widget подклассы) выбрасываются и перестраиваются при каждом изменении конфигурации.
- Поскольку State не сбрасывается при каждой перестройке, он позволяет избежать дорогостоящих вычислений и получает доступ к state свойству, геттерам, сеттерам и т. д. каждый раз, когда что-то перестраивается кадр за кадром.
- Важно то, что именно это позволяет существовать анимации Flutter. Так как State не выбрасывается, то может постоянно перестраиваться Widget в ответ на изменения данных, а также при необходимости.

1. createState()

- Когда Flutter получает указание создать StatefulWidget, он немедленно вызывает createState().

```
class MyHomePage extends StatefulWidget {
  @override
  _MyHomePageState createState() => new _MyHomePageState();
}
```

2. mounted is true

- При createState создании класса состояния buildContext этому состоянию присваивается.
- А BuildContext — это, в упрощенном виде, место в дереве виджетов, в котором этот виджет размещается
- У всех виджетов есть bool this.mounted свойство.
- Это свойство полезно, когда вызывается метод в состоянии, setState().

3. initState()

- Это первый метод, вызываемый при создании виджета (после конструктора класса).
- initState вызывается один и только один раз. Он также должен вызвать super.initState().

Этот @override метод:

- Инициализирует данные, основанные на конкретном BuildContext для созданного экземпляра виджета.
- Инициализирует свойства, которые зависят от «родителя» этого виджета в дереве.
- Streams, ChangeNotifiers или любой другой объект, который может изменить данные в этом виджете.

4. didChangeDependencies()

- Метод DidChangeDependities вызывается сразу после первой initState сборки виджета.
- Он также будет вызываться всякий раз, когда вызывается объект, от данных которого зависит этот виджет . Например, если он использует InheritedWidget, который обновляется.
- build всегда вызывается после didChangeDependencies вызова, поэтому это требуется редко.

5. build()

- Этот метод вызывается часто (например, fps + render). Это обязательный параметр, @override который должен возвращать Widget.

6. didUpdateWidget(Widget oldWidget)

- didUpdateWidget() вызывается, если родительский виджет изменяется и должен перестроить этот виджет (поскольку ему нужно передать ему другие данные), но он перестраивается с тем же самым runtimeType, то вызывается этот метод.
- Это связано с тем, что Flutter повторно использует метод state, который существует уже долгое время. В этом случае необходимо снова инициализировать некоторые данные, как это делается в initState().
- Если метод состояния build() зависит от Stream или другого объекта, который может измениться, отменить подписку на старый объект и повторно подписаться на новый экземпляр didUpdateWidget().
- Совет : Этот метод по сути является заменой initState(), если ожидается, что Widget связанные с виджетами будут перестроены!
- Flutter всегда вызывается build() после этого, поэтому любые последующие вызовы setState излишни.

7. setState ()

- Метод setState() часто вызывается из самой среды Flutter и от разработчика.
- Он используется для уведомления фреймворка о том, что «данные изменились», и виджет при этом build context должен быть пересобран.
- setState() принимает обратный вызов, который не может быть асинхронным . Именно по этой причине его можно вызывать часто по мере необходимости.

8. deactivate()

- Это используется редко.
- deactivate() вызывается при State удалении из дерева, но он может быть вставлен повторно до завершения текущего изменения кадра. Этот метод существует в основном потому, что State объекты можно перемещать из одной точки дерева в другую.

9. dispose()

- dispose() вызывается, когда State объект удаляется, что является постоянным.
- Этот метод позволяет отказаться от подписки и отменить все анимации, потоки и т. д.

10. mounted is false

- Объект state никогда не сможет перемонтироваться, и выдается ошибка setState().

Дерево виджетов

- В любой момент приложение Flutter будет состоять из множества виджетов, связанных между собой древовидной структурой. Это мало чем отличается от DOM в веб-браузере, который организует HTML-элементы в виде своего рода дерева. Дерево виджетов — это не только реальная структура данных, созданная платформой.
- Дерево виджетов строиться с помощью build метода в объектах виджетов. Каждый раз, когда метод сборки возвращает больше виджетов, все эти виджеты становятся узлами в дереве. Когда пользователь взаимодействует с приложением, Flutter использует эту древовидную структуру для представления виджетов приложения. Когда пользователь переходит на новый экран, Flutter удалит все виджеты в дереве, которые больше не используются (с экрана, с которого они перешли), и заменит их виджетами, представляющими новую страницу.

BuildContext()

- Каждый виджет Flutter имеет @override build() метод с аргументом BuildContext:
- расположение Widget в дереве виджетов.
- виджет виджетов, например вложенный.
- родительские объекты.
- У каждого Widget свое build() и свое context.
- BuildContext является родительским элементом виджета, возвращаемого build() методом.
- Другими словами, вызываемый виджет buildContext не совпадает с контекстом сборки виджета, возвращаемого.

Метод «of()»

- Во Flutter, как и везде, есть виджеты, просматривающие дерево виджетов вверх и вниз, в некоторых случаях для ссылки на другие виджеты. Это необходимо для некоторых функций.
- В частности, виджеты, которые хотят использовать состояние inherited виджетов, должны иметь возможность ссылаться на эти унаследованные виджеты. Обычно это проявляется в форме метода of.

Метод Builder

- Builder — это виджет, который принимает замыкание и использует его для создания дочерних виджетов. Его можно использовать для передачи контекста из метода build непосредственно дочерним элементам, возвращаемым в этом build методе.