Dart. Async

Dart — однопоточный язык

Для начала следует запомнить, что Dart — однопоточный а Flutter использует Dart.

Dart исполняет одновременно одну и только одну инструкцию. Последовательно. Это значит, что выполнение одной инструкции не может быть прервано другой инструкцией.

Другими словами, если мы рассматриваем синхронную функцию (или метод класса), то она будет единственной исполняемой прямо сейчас до тех пор, пока не завершится.

```
void myBigLoop(){
    for (int i = 0; i < 1000000; i++){
        _doSomethingSynchronously();
    }
}
```

Модель выполнения в Dart

Event Loop -Компонент Dart, который управляет очерёдностью исполнения инструкций.

Когда вы запускаете Flutter-приложение (и любое Dart-приложение), создается и запускается новый процесс — Thread, в терминах Дарта — Изолят (Isolate). Это единственный процесс, в котором будет выполняться ваше приложение.

Когда этот процесс создан, Дарт автоматически выполняет следующие действия:

- инициализирует две очереди (Queues) с именами MicroTask (микрозадания) и Event (событие), тип очередей FIFO (прим.: first in first out, т.е. сообщение, пришедшие раньше, будут раньше обработаны),
- исполняет метод main() и, по завершении этого метода
- запускает Event Loop (цикл событий)

```
void eventLoop(){
    while (microTaskQueue.isNotEmpty){
        fetchFirstMicroTaskFromQueue();
        executeThisMicroTask();
        return;
    }

    if (eventQueue.isNotEmpty){
        fetchFirstEventFromQueue();
        executeThisEventRelatedCode();
    }
}
```

Очередь MicroTask

Используется для очень коротких действий, которые должны быть выполнены асинхронно, сразу после завершения какой-либо инструкции перед тем, как передать управление обратно Event Loop.

Очередь Event

Используется для планирования операций, которые получают результат от:

внешних событий, таких как

операции ввода/вывода

жесты

рисование

таймеры

потоки

...

Future

Фактически, каждый раз при срабатывании внешнего события, соответствующий код ставится в очередь Event.

Futures

Future представляет собой задачу, которая выполняется асинхронно и завершается (успешно или с ошибкой) когда-то в будущем.

Что происходит, когда вы создаёте экземпляр Future:

- экземпляр создаётся и хранится во внутреннем массиве, управляемом Dart
- код, который должен быть исполнен данным экземпляром Future, добавляется напрямую в очередь Event
- возвращается экземпляр Future со статусом не завершено (incomplete)
- если есть синхронный код для исполнения, он исполняется (но не код Future)

```
void main(){
    print('Before the Future');
    Future((){
        print('Running the Future');
    }).then((_){
        print('Future is complete');
    });
    print('After the Future');
}
```

Before the Future

After the Future

Running the Future

Future is complete

Код Future НЕ выполняется параллельно, он выполняется в последовательности, определяемой Event Loop

Асинхронные методы (async)

Когда вы помечаете объявление метода ключевым словом async, для Dart это значит, что:

результат выполнение метода — это Future

он синхронно выполняет код этого метода, пока не встретит первое ключевое слово await, после чего исполнение метода приостанавливается

оставшийся код будет запущен на исполнение как только Future, связанный с ключевым словом await будет завершён

```
void main() async {
  methodA();
  await methodB();
  await methodC('main');
  methodD();
}

methodA(){
  print('A');
}

methodB() async {
  print('B start');
  await methodC('B');
  print('B end');
}

methodC(String from) async {
  print('C start from $from');

  Future((){              // <== Этот код будет исполнен когда-то в
будущем
    print('C running Future from $from');
  }).then((_){
    print('C end of Future from $from');
  });

  print('C end from $from');
}

methodD(){
  print('D');
}
```

A

B start

C start from B

C end from B

B end

C start from main

C end from main

D

C running Future from B

C end of Future from B

C running Future from main

C end of Future from main

Асинхронный метод НЕ выполняется параллельно, он выполняется в последовательности, определяемой Event Loop

```
void method1(){
  List<String> myArray = <String>['a','b','c'];
  print('before loop');
  myArray.forEach((String value) async {
    await delayedPrint(value);
  });
  print('end of loop');
}
```

```
void method2() async {
  List<String> myArray = <String>['a','b','c'];
  print('before loop');
  for(int i=0; i<myArray.length; i++) {
    await delayedPrint(myArray[i]);
  }
  print('end of loop');
}
```

```
Future<void> delayedPrint(String value) async {
  await Future.delayed(Duration(seconds: 1));
  print('delayedPrint: $value');
}
```

Многопоточность?

Isolates (Изоляты)

Isolate в Dart соответствует общепринятому Thread

Изоляты во Flutter не разделяют память. Взаимодействие между разными Изолятами реализовано посредством сообщений

Свой EventLoop

Свои MicroTask и Event

создание и рукопожатие

```
SendPort newIsolateSendPort;
Isolate newIsolate;
void callerCreateIsolate() async {
    ReceivePort receivePort = ReceivePort();
    newIsolate = await Isolate.spawn(
          callbackFunction,
          receivePort.sendPort,
       );
    newIsolateSendPort = await receivePort.first;
}

callbackFunction(SendPort callerSendPort){
 ReceivePort newIsolateReceivePort = ReceivePort();

 callerSendPort.send(newIsolateReceivePort.sendPort);
 }
```

отправка сообщения Изоляту

```
Future<String> sendReceive(String messageToBeSent) async {
   ReceivePort port = ReceivePort();
   newIsolateSendPort.send(
      CrossIsolatesMessage<String>(
         sender: port.sendPort,
         message: messageToBeSent,
      )
   );
   return port.first;
}
```

```
class CrossIsolatesMessage<T> {
   final SendPort sender;
   final T message;

   CrossIsolatesMessage({
      @required this.sender,
      this.message,
   });
}
```

```
static void callbackFunction(SendPort callerSendPort){
   ReceivePort newIsolateReceivePort = ReceivePort();
   callerSendPort.send(newIsolateReceivePort.sendPort);
   newIsolateReceivePort.listen((dynamic message){
      CrossIsolatesMessage incomingMessage = message as CrossIsolatesMessage;
      String newMessage = "complemented string " + incomingMessage.message;
      incomingMessage.sender.send(newMessage);
   });
}
```

удаление Изолята

```
void dispose(){
    newIsolate?.kill(priority: Isolate.immediate);
    newIsolate = null;
}
```