Корутины

Введение

Что такое синхронный код? - Когда следующая операция выполняется после завершения предыдущей сверху вниз
Преимущества синхронного подхода? - Легко писать, читать и поддерживать
Недостатки синхронного подхода? - Иногда логика кода пишется так, что не получится запустить весь код в одном потоке. Например нам нужно сделать 2 запроса в сеть из фонового потока, а между ними обновить UI из main потока.
Что такое асинхронный подход? - запуск какой-либо операции на отдельном фоновом потоке, по ее окончанию мы можем получить оповещение через callback
Преимущества асинхронного подхода? - можно использовать разные потоки для различных операций в одном блоке кода
Недостатки асинхронного подхода? - читать, писать и поддерживать такой код тяжело, особенно при большом кол-ве операций
Что такое CallbackHell? - огромная вложенность коллбэков, сложная для понимания
Для чего нужны корутины? - Они позволяют писать асинхронные программы, которые будут выглядеть, как синхронные. Соответственно, корутины объединяют в себе плюсы обоих подходов: простота понимания, написания и поддерживания, а также возможность использования асинхронных операций
Что означает модификатор suspend? - Функция может приостановить свое выполнение и в нужный момент продолжить его, не блокируя при этом поток
Как работает suspend изнутри? - По сути, просто под капотом преобразовывает наш код в код с функциями с коллбэками. Подробнее см. раздел “Как работает suspend”
Что такое корутины? - Легковесные потоки, которые под капотом работают на тех же самых потоках. На одном потоке может быть запущено несколько корутин одновременно

Преимущества корутин? -

1) Легковесные (1 поток занимает ~1 Мб оперативной памяти, а корутина несколько десятков байт)
2) Эффективные (приостанавливают поток и не занимают его ожиданием результата)
3) Позволяют избежать утечек памяти с помощи привязки к определенному ЖЦ компонента
4) Простая отмена
5) Обработка ошибок
6) Интегрированы с компонентами Jetpack

Работа с корутинами

Как может быть вызвана suspend функция? - только из другой suspend функции или из корутины

Для чего нужен Coroutine Builder? - позволяет создать и запустить корутину. По сути, Coroutine Builder - это обычные функции

Какие существуют Coroutine Builder'ы? -

1) runBlocking - блокирующе запускает корутину в текущем потоке. Используется в основном для тестов с корутинами
2) launch - запускает корутину на выполнение
3) async - запускает корутину и позволяет получить результат по окончании выполнения

Что такое Structured Concurrency? - парадигма, используемая при работе с корутинами, согласно которой мы должны понимать:

1) кто может отменить выполнение корутины?
2) привязано ли выполнение к какому-то ЖЦ?
3) как будут обрабатываться ошибки внутри корутины?
4) как корутины взаимодействуют с дочерними или с родительскими корутинами

Для чего нужен Coroutine scope объект? -

1) позволяет отслеживать выполняемые в нем корутины
2) позволяет отменить выполнение всех его корутин
3) оповещается при возникновении ошибки внутри корутины
4) в одном скоупе может быть разное количество корутин

Как можно настраивать Coroutine scope? -

1) определять поток для корутин

2) поведение при ошибках
3) отменять корутины

Как создать объект Coroutine scope? -
val scope = *CoroutineScope*() - функция принимает в себя coroutineContext, который обязательно имеет Job'у(можно не указывать, если нас устраивает дефолтная Job'а, она создается по умолчанию), и дополнительно можно передать Dispetcher для указания потока, в котором будет работать корутина по умолчанию и обработчик ошибок CoroutineExceptionHandler.
Для чего нужен метод withContext()? - позволяет изменить диспетчер для какого-либо блока кода
Какие существуют диспетчеры? -

1) Default - количество потоков = кол-ву ядер. Используется для ресурсоемких задач
2) IO - max кол-во потоков = 64. Используется для задач ввода/вывода. Переиспользует потоки Default диспетчера
3) Main - выполняет задачи на главном потоке
4) Unconfined - не переключается между потоками
5) Custom - можно создать диспетчер самостоятельно

Как работает метод async? - возвращает результат в качестве объекта Deferred, у которого можно получить, например, статус состояния корутины
Как работать с методом async? - чтобы получить результат выполнения корутины нужно дождаться ее завершения с помощью метода await(). В свою очередь, так как await() является suspend функцией его необходимо тоже вызывать в корутине. Следовательно нужно создать родительскую корутину, вызвать там корутину async, и затем использовать в родительской корутине метод await()

Как работает suspend?

Как андроид обрабатывает suspend функции? - т.к. Андроид имеет внутри виртуальную машину, которая ничего не знает о саспенд функциях и о корутинах, компилятор при переводе в байткод избавляется от саспендов в пользу оптимизированной версии коллбэков с использованием стейт машины. В этом ему помогает интерфейс Continuation
Что такое интерфейс Continuation? - интерфейс, который позволяет нам продолжить выполнение после точки остановки. Внутри имеет контекст

корутины и метод resume, который позволяет продолжить выполнение приостановленной функции

Как работает suspend изнутри? - при компиляции suspend функция преобразуется в объект класса Continuation, который имеет метод invokeSuspend, внутри которого и происходит основная логика. Код корутины преобразовывается в стейт машину и разбивается на несколько состояний в соответствии с точками приостановки

Что такое стейт машина? - набор состояний и определенные переходы, которые связывают состояния между собой

Как преобразовать код с коллбэками в саспенд функцию, если корутины не поддерживаются нашим API по умолчанию? - с помощью функции suspendCoroutine, которая передаст в лямбду объект continuation. В эту функцию вставляем наши коллбэки, а в качестве их реализации используем continuation.resume(полученный результат) или continuation.resumeWithException(полученная ошибка). Также необходимо сменить поток на фоновый, если библиотека не делает это сама.

CoroutineScope

Что такое интерфейс Job? - интерфейс, который определяет состояние корутины в определенный момент времени

Как получить инстанс Job'а? - используя CoroutineBuilder

Что можно сделать с помощью job'а? - получить состояние корутины, отменить корутину, подождать окончания корутины

Как устроены отношения дочерних и родительских корутин? - если завершается родительская корутина, то завершаются и все дочерние. Если завершается дочерняя, то родительская не завершается. Перед завершением родительской корутины всегда сначала должна завершится дочерняя корутина

Что такое CoroutineScope? - интерфейс, инстанс которого внутри себя хранит CoroutineContext(по сути, просто мапа), который содержит Job, Dispetcher, CoroutineExceptionHandler и объект, имплементирующий CoroutineContext.Element (можно положить любой объект, нужно например, чтобы объекты были доступны из нескольких корутин)

Как работает метод cancel у scope? - достается Job'а , и уже у нее вызывается cancel()

Отмена, обработка ошибок в корутинах

Самый простой способ обработки ошибок в корутинах? - обернуть код внутри корутины в try/catch

Как обрабатывать ошибки внутри корутины с помощью CoroutineExceptionHandler? - необходимо создать объект CoroutineExceptionHandler, который принимает на вход лямбду, которая будет вызывать в тот момент, когда произошла ошибка. Затем добавить этот объект в coroutineContext

Что происходит в корутине при возникновении ошибки? -

1) Continuation возвращает результат ошибки
2) Оповещается родительский job
3) Родительский job отменяет дочерние корутины
4) В CoroutineContext у scope проверяется наличие exception handler
5) Обработка исключения/краш приложения, scope отменяется

Как по умолчанию ведет себя Job'а при возникновении ошибки в одной из дочерних корутин? - отменяет себя и все дочерние корутины, даже те, в которых ошибки не было и даже если ошибка была обработана

Что такое SupervisorJob? - это Job'а, которая при возникновении ошибки в дочерней job'е не отменяет родительскую job'у

Что такое CancellationException? - специальное исключение, возникающее внутри корутины в момент отмены. Не распространяется вверх, как обычное исключение. Распространяется на дочерние корутины

Как отменить корутину? - можно через метод cancel(), но это не лучшее решение, так как после этого скоуп будет завершен и мы не сможем больше запустить другие корутины. Лучше использовать cancelChildren().

Как обработать отмену корутины внутри корутины, построенной на коллбэках? -

1) использовать метод suspendCancellableCoroutine внутри CoroutineBuilder'а. В нем использовать у переданного continuation.invokeOnCancellation{}, внутри которого сделать что-то, что позволит отменить код в корутине
2) проверять вручную статус корутины isActive
3) использовать метод yield

Что такое метод yield? - метод, запрашивающий состояние корутины и отменяющий дальнейшее его выполнение, если корутина была отменена

Корутины в Андроид

Что такое интерфейс lifecycleOwner? - интерфейс, который имплементируется фрагментом. Сущность, которая имеет жизненный цикл. Фрагмент имеет ЖЦ, который, соответственно, является lifecycleOwner

Что такое lifecycleScope? - extension свойство lifecycleOwner'a. CoroutineScope, привязанный к ЖЦ lifecycleOwner'a. Использует SupervisorJob по умолчанию. Можно передать диспетчер или обработчик ошибок при желании

Что такое viewLifecycleOwner и viewLifecycleScope, а также viewModelScope? - тоже самое, что и lifecycleOwner и lifecycleScope, только относительно view и ViewModel,соответственно

Best Practices

1) Оборачивать асинхронные вызовы (async) в coroutineScope (предпочтительнее) или использовать SupervisorJob для обработки исключений во избежании краша приложения из-за ошибки в блоке async
2) При использовании корневых корутин в большинстве случаев предпочтительнее создать CoroutineScope сразу с главным диспетчером, что приведёт к упрощению кода и менее явному переключению контекста.
3) Избегать использования ненужных async/await. Если необходимо переключить контекст корутины и немедленно приостановить родительскую корутину, то withContext — это самый предпочтительный для этого способ. С точки зрения производительности это не такая большая проблема (даже если учесть, что async создаёт новую корутину для выполнения работы), но семантически async подразумевает, что вы хотите запустить несколько корутин в фоновом режиме и только потом ждать их.
4) Избегать ручной отмены job, так как эта завершает scope, и мы не сможем открыть другие корутины. Если мы хотим отменить все корутины в определенной области, вы можете использовать функцию cancelChildren. Кроме того, хорошей практикой является предоставление возможности отмены отдельных job

5) Не использовать GlobalScope. Лучше использовать scope, привязанный к ЖЦ элементов Андроид