Coroutines

From request to Ui-state

ДВФУ · ИМКТ · Департамент МиКМ · Android-разработка · 2025

Agenda

Корутины + сеть

UI -> ViewModel -> Repository -> Api

Без Flow*

Что мы знаем?

Compose UI

ViewModel + UiState

Navigation

следующий шаг - **данные**

HTTP

HTTP: что это вообще?

- HTTP = протокол **клиент - сервер**
- Модель: **Request - Response**
- Обычно поверз HTTPS (HTTP + шифрование)

Из чего состоит HTTP Request

- Method: GET / POST / PUT / DELETE …
- URL: scheme://host/path?query
- Headers (метаданные)
- Body (данные; чаще у POST/PUT)

HTTP-методы

Метод	Классический Web (HTML + Form Submit)	REST Services AJAX
GET	Получение страниц	Получение сушности
POST	Отправка данных формы	Создание сушности
PUT	-	Изменение сушности (замена)
PATCH	-	Изменение сушности (частичное)
DELETE	-	Удаление сушности
HEAD	ХИТРЫЙ	ХИТРЫЙ
OPTIONS	CORS	ХИТРЫЙ
TRACE	НЕ ИСПОЛЬЗУЕТСЯ	НЕ ИСПОЛЬЗУЕТСЯ

HTTP-коды

- 1xx – информационные (протокол HTTP, не увидите)
- 2xx - всё OK
- 3xx – редирект
- 4xx - ошибка, виноват клиент
- 5xx - ошибка, виноват сервер

HTTP

```
curl https://rickandmortyapi.com/api/character/47 -v

GET /api/character/47 HTTP/1.1
Host: rickandmortyapi.com
User-Agent: curl/8.9.1
Accept: */*


HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Age: 0
Cache-Control: max-age=259200
Cache-Status: "Netlify Edge"; fwd=miss
Content-Length: 757
Content-Type: application/json; charset=utf-8
Date: Sat, 30 Nov 2024 14:32:51 GMT
Etag: W/"2f5-Bx1QWKnwBYBYiLEbTvzQgnL6CzM"
Expires: Tue, 03 Dec 2024 14:32:51 GMT
Netlify-Vary: query
Server: Netlify
Strict-Transport-Security: max-age=31536000
X-Nf-Request-Id: 01JDYQT6NN4390MQCT8EERZVDD
X-Powered-By: Express

{"id":47,"name":"Birdperson",...}
```

Что важно помнить про HTTP

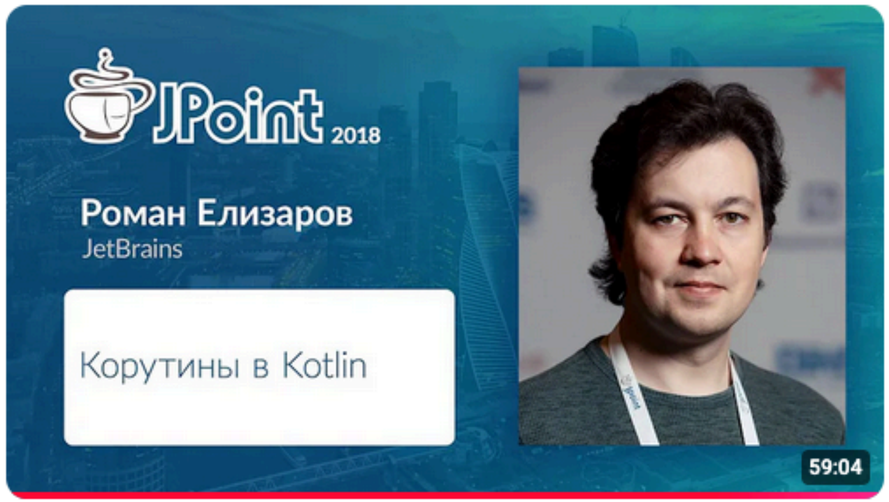
- Протокол, это договор/контракт
- HTTP, это текстовый протокол. Договоренность об обмене текстовыми блоками
- Строится на «запрос-ответ»
- Есть ряд HTTP-методов для запросов (request)
- Есть ряд статус-кодов для ответов (response)

Сеть “на практике”: задержка, таймаут, повтор

- Сеть медленная / нестабильная
- Возможные проблемы:
  - нет интернета (DNS)
  - таймаут
  - временная ошибка сервера
- UX-минимум:
  - Loading
  - Error + Retry
  - не делать запрос на каждый символ

Coroutines

Автор



Роман Елизаров — Корутины в Kotlin

72 тыс. просмотров • 7 лет назад

JPoint, Joker и JUG ru — Java-конференции

Подробнее о Java-конференциях: — весной — JPoint: http

KotlinConf 2017 — Введ

131 тыс. просмотров • 8 лет назад

JetBrains

Мы живём в эпоху асинхронного пара

Субтитры Английский

Эпизоды (15) Introductio

KotlinConf 2018 - Сопрограммы Kotlin на практике, Р

93 тыс. просмотров • 7 лет назад

JetBrains

Запись предоставлена вам American Express https://americanexpress.io/kotlin-jobs Д

Субтитры Английский

Моменты (3) Shared + Mutable | Communication Primitives | Coroutines a

JPoint, Joker и JUG ru — Java-конференции

Подробнее о Java-конференциях: — весной — JPoint: https://jrg.su/gTrwHx — осенью — Joker: http

Проблема: «долгая операция» и UI

- Сеть/диск/парсинг могут занимать сотни мс и секунды
- Если делать это на main - UI зависает
- Нужно асинхронно

Что такое корутина

- Корутина = лёгкая “приостанавливаемая” вычислительная задача
- Пишем асинхронный код “как последовательный”

- Строится на **suspend**-функциях
- suspend позволяет “приостановить/продолжить” без блокировки потока

- *корутина = “легковесный поток”

suspend: почему функция не возвращает сразу

- suspend fun может “подождать” без блокировки потока
- Важно: suspend ≠ новый поток

```
suspend fun foo() {

}
```

Structured concurrency (идея)

- Запускаем **работу** в **scope**
- Когда scope “умирает” - работа отменяется
- Меньше утечек и зависших задач

Android way: ViewModel + viewModelScope

- В Android обычно:
  - UI -> вызывает событие (event)
  - ViewModel -> запускает coroutine и меняет UiState

viewModelScope: где запускать сеть

- viewModelScope.launch { ... }
- Корутины автоматически отменяются, когда ViewModel очищается

https://developer.android.com/topic/libraries/architecture/coroutines

Диспетчеры: где выполняется работа

- Dispatchers.Main - UI
- Dispatchers.IO - сеть/диск
- Dispatchers.Default - CPU

Coroutines builders

Start new coroutine

- **launch** - “fire-and-forget”, возвращает Job
- **async** - возвращает Deferred<T> (результат через await())

```
scope.launch { ... }
scope.async { ... }
```

Переключатели контекста

меняют “где/как” выполняется текущая корутина

- withContext(Dispatchers.Default) { ... } - переключить и дождаться результата (поэтому suspend)
- coroutineScope { ... } - создать дочерний scope и дождаться всех детей (structured concurrency)
- supervisorScope { ... } - как coroutineScope, но с supervisor-поведением ошибок

Scope

- CoroutineScope(context) - создаёт scope вручную
- готовые scope в фреймворках: viewModelScope

```
val scope = CoroutineScope(Job() + Dispatchers.Default)

// ...

scope.cancel()
```

Context

набор “параметров” корутины

Задача: описать поведение выполнения.

Элементы контекста:

- Dispatchers.Default / IO / Main / Unconfined
- Job / SupervisorJob
- CoroutineName
- CoroutineExceptionHandler
- и т.д.

Context

Example

```
val ctx = SupervisorJob() + Dispatchers.Default + CoroutineName("worker")
val scope = CoroutineScope(ctx)
```

Варианты Джобов

Обычный **Job (по умолчанию)**

- Если у родителя обычный Job, то падение одного ребёнка отменяет родителя, а значит обычно отменяются и остальные дети.

SupervisorJob

- С SupervisorJob() падение одного ребёнка не отменяет родителя и не валит остальных детей.

Мини архитектура

- ApiService (Retrofit)
- Repository
- ViewModel (UiState + events)
- UI (stateless)