Reactive Streams using Spring Webflux 6 Bogdan Sucaciu

A close-up of a webflux

Description automatically generated

Reactive vs Imperative Programming

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Imperative

-Code executions gets blocked until we get the data we requested

Reactive

-Code execution will continue even when dealing with blocking operation

-Data comes in as streams (piece by piece) instead of blocks (one big chunk)

-Functional:

-asynchronous code can be implemented either with callbacks or futures but Functional change imperative blocks of codes into functions and chain them to the desired result

A purple and white rectangular object with text

Description automatically generated

Harddrive, I/O includes DB and network calls

Reactive Streams

Reactive Programming

-Set of principles with no standards or implementation

Reactive Streams

-Standards are defined by this

A diagram of a company

Description automatically generated

Components

Publisher

-code providing the data under the form of data stream

Subscriber

-Consumes data and process it

A diagram of different types of software

Description automatically generatedSubscription

-Connect to publisher and receive data under the form of data stream

Processor

-Combination?

Project Reactor

-Library that implements all reactive streams specifications

A blue and orange rectangles with white text

Description automatically generated

Flux: represents a flux of data

Mono: one data is sent

A black text with orange and black text

Description automatically generated

-In case something bad happens, we can handle it gracefully

A diagram of a web application

Description automatically generated

-MVC framework is based on Servlet API (old specification that provides web capabilities in blocking fashion)

-Spring wanted non blocking capabilities so it made WebFlux (stack?).

-WebFlux is using Project Reactor under the hood and using Reactive Streams specification and not the Servlet API

A diagram of a flowchart

Description automatically generated

-The need came from Microservices

Project Loom

A diagram of gears and a diagram of a diagram

Description automatically generated

MongoDb using Docker



A screenshot of a computer code

Description automatically generated

-just method accepts any object as input

A screenshot of a computer

Description automatically generated

-Think of range as for loop

Spring MVC is Apache Tomcat but started Netty?

Exploring Project Reactor

A screenshot of a computer

Description automatically generated

A close up of a black background

Description automatically generated

-Use Spring Data JPA Library with Spring Data R2DBC (Reactive Relational DB Connectivity) for relational DB

A black background with white text

Description automatically generated

-Switch to these driver

Map

DTO

-Basically we don’t want to expose our internal data structure



-MethodReference instead of lambda expression

A screen shot of a computer program

Description automatically generatedDTO

StockRequest.java

A screen shot of a computer code

Description automatically generated

StockResponse.java

A screen shot of a computer

Description automatically generated

Error Handling

A screenshot of a computer

Description automatically generated

-Can no longer use try catch to transform and recover from exceptions

A diagram of error

Description automatically generated

oOnErrorReturn: provides a fallback object

onErrorResume: similar but takes exception as lambda argument

onErrorMap: transform one exception to another

A close up of words

Description automatically generated

-Wont work because the exception is thrown during the reactive pipeline

A screenshot of a computer code

Description automatically generated

-just sends through a reactive pipeline

-flatmap: Unwrapping Mono/Flux from its arguments

Example:

-stocksRepository.save(stock) returns a Mono, if we just a map it would return Mono<Mono<>>

Peek

A diagram of a program

Description automatically generated

doFirst: called before subscription

doOnSubscribe: immediately after subscriber initiates a subscription

doOnRequest: when sending a request

doOnNext: inspects data items published on stream

doOnTerminate: executes every time stream ends (completion or error) executes before

doAfterTerminate: executes after last element is propagated to stream

Using WebClient

-Probably use REST template for API calls, won’t work for reactive paradigm

-WebClient replaces Spring REST template

A screenshot of a computer program

Description automatically generatedA diagram of a service

Description automatically generated

-A popular structure

Client

-makes API calls to other apps

A close-up of a purple square

Description automatically generated

-retrieve triggers the request

A diagram of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

-Use the flatmap if inner lambda expressions returns a mono

-singleOrEmpty(): transfer a Flux into a Mono because we only expect one

A computer screen shot of a program code

Description automatically generated

@Transactional

-The save to the DB will be rolled back if an error occurs

A screenshot of a computer

Description automatically generated

-Order matters

A screenshot of a computer

Description automatically generated

-Difference between bodyToMono and createException is that we can pass more information using a DTO

-Chosing one may come down to the upstream app is sending an usable response body

A screenshot of a computer

Description automatically generated

-exchange gives us access to the entire response

- a lot more flexibility

WebClient Filters

-