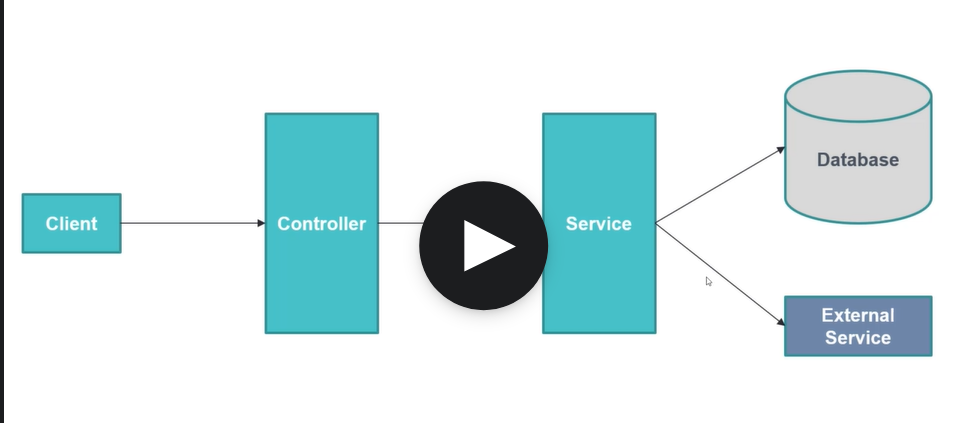
Order order=orderRepository.findById(id).get();

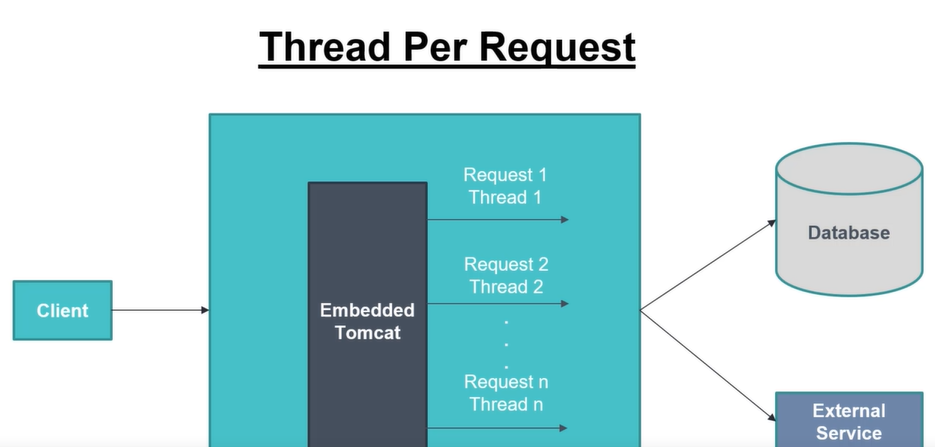
CourierDetails courierDetails=restTemplate.getForObject()

Pehle first step mein 5 sec wait krega then second step mein wait rega

Its a synchronized in a blocking way



Yha pe app pehle db ko call krega then service ko call krega



Ek thread db ko bhej kar idle ho jati hai

Then dusri thread external service ko bhej kr ideal ho jati hai fir jab dono ka resp aa jata hai to dono club hokr client ko response jata hai

Thread no is finite

BackPressure:

Spring boot app–findAll()---DB

back pressure ko handle krne ke liye ko way ni hai traditional Rest API mein’

Challenges of traditional REST api

1.synchronous and blocking

2.thread per request model

3.no back pressure

What is reactive programming?

Declarative programming approach

Async n non blocking

Data flow happens with the help of event driven streams

Supports back pressure

Functional style coding

Event driven stream:

Data flow happens with the help of events

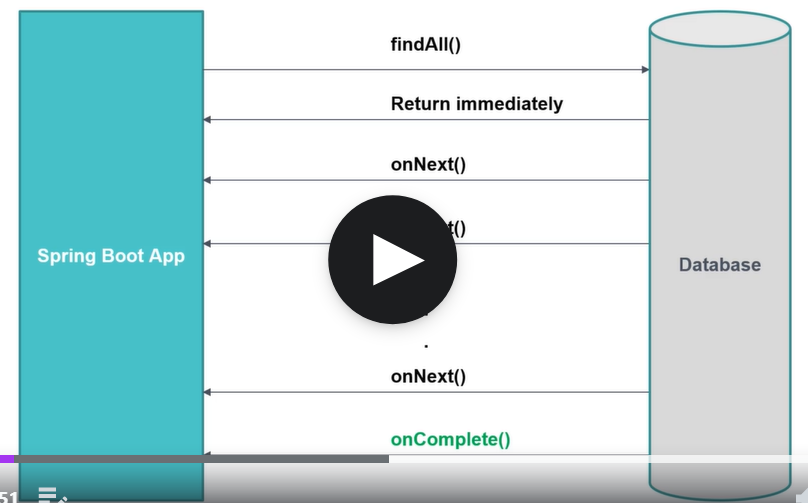
What is events?

Each data received from data source as events

Compilation or Error

Publisher provides data

Subscriber receives data



Reactive programming mein response turant return ho jata hai

Har cheej events ke through hi hogi via onNext()

Failure scenario::

What us reactive stream:

Rules for reactive stream…Reactive Stream Specification:

It has four specification

1.publisher db

Public interface Publisher<T>{ OTT

Public void subscribe(Subscriber<? Super T>s); ME

}

Subscriber will call subscribe to subscribe to method

2.subscriber client who receives the data

Public interface Subscriber<T>{

Public void onSubscribe(Subscription s);

Public void onNext(T t);

Public void onError(Throwable t);

Public void onComplete();

}

3.subscription.

It represents one to one relation between publisher and subscriber

Public interface subscription{

Public void request(long n)

Public void cancel();

}

4.Processor

vRepresents processing stage for processor and subscriber

Follows contract of both publisher and subscriber

Public interface Processor<T,R> extends Subscriber<T>,Publisher<R>{

}



Why Reactive programming?

Asynchronous and non blocking

Supports back pressure

REACTIVE libraries:

Project Reactor

RxJava

JDK 9 reactive streams

Introduction to FLUX and MONO

Theya re reactive typoes in reacti ve programming

FLUX represents multiple data

MONO represents single data

Both can be empty

[ INFO] (main) onSubscribe(FluxConcatArray.ConcatArraySubscriber)

[ INFO] (main) request(unbounded)

[ INFO] (main) onNext(John)

John

[ INFO] (main) onNext(Raj)

Raj

[ INFO] (main) onNext(Sachin)

Sachin

[ INFO] (main) onNext(peter)

peter

[ INFO] (main) onComplete()

Flux<String> stringFlux = Flux.just("John", "Raj", "Sachin").concatWith(Flux.just("peter")).log(); stringFlux.subscribe(System.out::println);

Flux<String> stringFlux = Flux.just("John", "Raj", "Sachin")

.concatWith(Flux.error(new RuntimeException()))

.concatWith(Flux.just("peter")).log();

stringFlux.subscribe(System.out::println,

e->System.out.println("Exception"),

()->System.out.println("success"));

//System.out.println(stringFlux);

Koi activity jo ki error ase mein krni ho to yha kr sakte h also same is for autocomplete

Understanding mono with examples:

Mono<String>stringMono=Mono.just("john").log();

//its only for the single value it acts as a publisher

stringMono.subscribe(System.out::println);

We will not get mono with null values

Mono<?>emptyMono=Mono.justOrEmpty(str).log();

emptyMono.subscribe();

//list to flux

List<String>list=Arrays.asList("anup","kill","pill");

Flux<String>stringFlux=Flux.fromIterable(list).log();

stringFlux.subscribe(System.out::println);

Flux from streams

List<String>list=Arrays.asList("anup","kill","pill");

Stream<String>stream=list.stream();

Flux<String>stringFlux=Flux.fromStream(stream);

stringFlux.subscribe(System.out::println);

//Creating FLUX with range

Flux<Integer>flux=Flux.range(5, 10).log();

flux.subscribe(System.out::println);

10 is count 5 is start

//apply filter on flux

Flux<String>flux=Flux.just("pilli","kuu","kuku","tim")

.filter(s->s.length()==3)

.log();

flux.subscribe(System.out::println);

//map operation with flux

Flux<Order>order=Flux.just(

new Order(1L,100.9),

new Order(2L,200.1),

new Order(3L,200.9)

);

Flux<Long>fluxLong=order

.filter(os -> os.getAmount()>100)

.map(orde->orde.getId()).log();

fluxLong.subscribe(System.out::println);

Flat map with FLUX

—------------------------------------------------------------------------------------------

Revisit back back pressure

Two method are imp request and cancel

intFlux.subscribe(new BaseSubscriber<Integer>() {

protected void hookOnSubscribe(Subscription subscription){

subscription.request(2);

}

protected void hookOnNext(Integer value){

System.out.println(value);

}

protected void hookOnComplete() {

System.out.println("Completed");

}

protected void hookOnError(Throwable throwable) {

throwable.printStackTrace();

}

});

Cancel method()

<<Introduction to schedulers>>

Schedulers are used to perform task asynchronously

Reactor provides scheduler

Immediate scheduler

Parallel scheduler we have pool of threads

Single scheduler

BoundedElastic scheduler for db call

**ImmediateScheduler**

**Flux<Long>fluxLong=orderFlux**

.publishOn(Schedulers.immediate())

**.map(**

**order->{**

**System.out.println("Map with "+Thread.currentThread().getName());**

**return order.getId();**

**}**

**);**

**fluxLong.subscribe(orderId-> System.out.println("Sub with "+**

**Thread.currentThread().getName()));**

**Parallel Scheduler**

**Flux<Long>fluxLong=orderFlux**

**.publishOn(Schedulers.parallel())**

**.map(**

**Started with main**

**Map with parallel-1**

**Sub with parallel-1**

**Map with parallel-1**

**Sub with parallel-1**

**Publish on dusri thread mein hoga**

**Isse pehle vala main thread mein hoga**

**Flux with single scheduler**

**Immediate main thread pr run hota hai**

**Single a separate single thread**

**Parallel everytime a new thread**

**Bounded Elastic Scheduler:**

**Its switching to a new thread**

**Bounded elastic can increase the thread pool size dynamically**

**Elastic is deprecated**

**How to add delay while publishing elements to the flux**

**Flux<Long>flux=Flux.just(1L,2L,3L).delayElements(Duration.ofSeconds(1));**

**flux.subscribe(System.out::println);**

**Here flux internally uses parallel scheduler its aFlux with delay**

**Reactive rest api with spring webFlux**

**Video streaming example**

**Traditional rest mein agar beech mein user ne stop kr diya hai fir bhi serverprocess krta rehta hai**

**Flux based reactive mein usi samay cancel event fire ho jata hai**

**Reactive REST API with mono**

**Introduction to Functional Web**

**We follow functional style of coding**

**Functional end points**

**1.Router function**

**2.Handler Function**

**Spring data R2DBC**

**@Repository**

**public interface OrderRepository extends R2dbcRepository<Order, Long>{**

**—----------------------------------------------------------------------------------------------------------------------------**

@Bean

public RouterFunction<ServerResponse> orderRoutes() {

return RouterFunctions

.route(RequestPredicates.GET("/api/order/getAll")

,

orderHandler::getAll)

.andRoute(RequestPredicates.GET("/api/order/getOne")

,

orderHandler::getOne);

}

Spring R2DBC reactive way to interact to database

Its under spring data and useful to connect to database

**—----------------------------------------------------------------------------------------------------------------------------**

Why do we need yet another programming model?