Spring boot webflux (Spring boot Reactive Programming)

==========:

=> Web Application is developed using MVC in Spring Boot.

==========:

=> Every request is a thread internally(servlets concept). that may communicate with DB, until it gets data from DB it(Request Thread) is in blocking Mode.

=> Here, this thread can not be assigned to any other request until it gives response back.

In Traditional web applications and Restful applications

=> Reactive Programming makes our application Resource utilization insted of Keeping Threads in idle mode, they are utilized properly.

=> It is implemented using Event-Loop, that indicates, when thread want to make Db call, it will provide SQL query to Event Loop and becomes Free. Now, DB call made to Database and in parallel thread can process other request.

=> Once, DB processing is done and received response then either same thread or another thread is allocated for response. Some time even it has to wait if all threads are busy.

Traditional web application (problem)

Server

(for 3 mins) Sitting silent

For Spring Boot apps

2 mins

(Blocking State)

Client (browser)

Request

1 req = 1 woker thread

SQL

•

For Android apps

Reactive Programming implementation in java

==========

-

==========

Use Project Reactor (WebFlux).

→ Use RxJava.

→

2 min

•

process

For high-performance microservices

Data

1 min

•

For distributed systems

-

Response

Tomcat

←

Use Mutiny (Quarkus) or Vert.x.

Use Akka Streams.

In Spring/Spring boot Programming. we use following technologies for normal web applications development

a) web MVC :: For Web application Development

b) Rest (MVC++) :: For Distributed App Development

c) Cloud :: For developing microservices

Netty Server / 3.1+

Reactive web application (Solution)

Spring Web Flux

Request #1 (a1)

(Netty Server)

(Free) (a4)

1 Worker Thread

(a2) (a6)

Reactive Process Event Loop

-Spring Reactive DB

2 mins

(a3)

DB Call (a4-parellel)

nd

2 Worker Thread

Request#2

(a5)

(a7-parellel)

Response#1

(a8-parellel)

Another name for spring web flux module is Project Reactor or spring Reactive or Spring Reactive web

1 min

(a6-parellel)

Data

DB s/w (a5-parelle)

or spring boot webflux module

=> At Server side, to implement application we should use Spring Web Flux (Spring 5.x). At Database side We should use Spring Boot Reactive Database(as of Now NoSQL DB only Supported).

Spring MVC

(MongoDB is Reactive DB) (like oracle 12c)

=> Insted of Spring REST applications Spring Reactive is Recomanded in future.

(already landed)

=> This is added as one Future plan as https://projectreactor.io/

note:: The Latest versions of Db s/ws likle

=> For Reactive Programming: Spring uses Netty Server, latest version of Servlets 3.1+

1. Spring ReST -> ResponseEntity as returnType (for endpoint method)

2. Spring Reactive /Spring boot Reactive

-> Mono(single object), Flux (multiple objects) as returnType (for endpoint method)

3. Other Annotations are similer: @RestController, @GetMapping..etc

4. RestTemplate(C) is client code for making HTTP calls to RestController

Bu this time, WebClient to make HTTP calls as Client Application

to our Reactive Programing.

5. Use Dependencies:

Spring Reactive Web (Web Flux), Spring Boot Ractive MongoDB, Lombok

Tradtional approach says the the communication b/w client and web application is synchronous communication i.e until given request response is gathered the same client can not generate another request or current request related thread can not take another request

oracle supporting the reactive programming ReactJS is also given based on Reactive Programming

(as a frond end technology)

Reactive Approach says the communication b/w

client and web application is asynchronous communication i.e the client can give next request with out waiting for given request related response from the web application

Using RestTemplate we can make consumer App taking to Rest API only in synchronous mode Using WebClient we can make consumer App taking to Rest API either in synchronous mode or asynchronous mode

Provider App Development (Spring webflux based Spring rest provider App)

1. Depen: Spring reactive web, Spring Data Reactive Mongo DB, Lombok

//Docu ment class

```
package com.nt.document;

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.mapping.Document;
```

import lombok.Data;

@Document(collection = "WEBFLUX_ACTOR_INFO")

@Data

public class ActorInfo {

@Id

private Integer aid;

private String aname;

private String category;

private Double fee;

Note:: Oracle, MySQL can be used for both traditional programming and also for reactive programming

Spring Reactive web module= Spring webflux module

BootWebFluxProj01-ReactiveProviderApp [boot] [devto

>

src/main/java

#com.nt

com.nt.advice

ActorOperationsControllerAdvice.java ActorOperationsControllerAdvice

⚫ handleAllExceptions(Exception): Respon

• handleRuntimeException(RuntimeExcept

com.nt.document

>

>

#com.nt.repository

}

3. Repository

package com.nt.repository;

import org.springframework.data.mongodb.repository.Reactive MongoRepository;

import com.nt.document.ActorInfo;

public interface IActorInfoRepository extends Reactive MongoRepository<ActorInfo, Integer> {

}

4.Service Interface

public interface IActor MgmtService {

public Mono<ActorInfo> saveActor(ActorInfo info); public Flux<ActorInfo> showAllActors();

public Mono<ActorInfo> showActorById (Integer id); public Mono<Void> removeActorById(Integer id); public Mono<ActorInfo> updateActor(ActorInfo actor);

5. Service Impl class

@Service

public class ActorMgmtServiceImpl implements IActorMgmtService {

@Autowired

private IActorInfoRepository actorRepo;

@Override

public Mono<ActorInfo> saveActor(ActorInfo info) {

//RestController Advice class package com.nt.advice;

>

>

>

> com.nt.rest

com.nt.service

src/main/resources

src/test/java

JRE System Library [JavaSE-17]

Maven Dependencies

target/generated-sources/annotations

target/generated-test-sources/test-annotations

>

>

>

>

src

>

target

HELP.md

mvnw

mvnw.cmd

Mpom.xml

import org.springframework.http.HttpStatus; import org.springframework.http.ResponseEntity; import org.springframework.web.bind.annotation.ExceptionHandler; import org.springframework.web.bind.annotation.RestControllerAdvice;

@RestControllerAdvice

public class ActorOperationsControllerAdvice {

@ExceptionHandler(RuntimeException.class)

public ResponseEntity<String> handleRuntimeException (RuntimeException re){

}

return new ResponseEntity<String>("Internal problem:"+re.getMessage(),HttpStatus.INTERNAL_SERVER_ERROR);

```
Mono<ActorInfo> savedDoc-actorRepo.save(info); return savedDoc;
}
@Override
public Flux<ActorInfo> showAllActors() {
return actorRepo.findAll().switchIfEmpty(Flux.empty());
}
@Override
public Mono<ActorInfo> showActorById(Integer id) {
try {
Thread.sleep(100000);
}
catch(Exception e) {
e.printStackTrace();
}
return actorRepo.findById(id).switchIfEmpty(Mono.empty());
}
@Override
public Mono<Void> removeActorById(Integer id) {
//Load the docuemnt
Mono<ActorInfo> mono-actorRepo.findById(id);
//delete the document
return mono.flatMap(obj->{
return
actorRepo.delete(obj).then();
})
.onErrorResume(e->{
return Mono.error(new RuntimeException("ActorInfo Document did not found for deletion",e));
});
}
*/
@Override
public Mono<Void> removeActorById(Integer id) {
return actorRepo.findById(id)
switchIfEmpty(Mono.error(new RuntimeException("ActorInfo document not found for deletion")))
.flatMap(actorRepo::delete)
.then();
}
```

```
@Override
public Mono<ActorInfo> updateActor(ActorInfo actor) {
Mono<ActorInfo> mono-actorRepo.findById (actor.getAid());
return mono.flatMap(obj->{
BeanUtils.copyProperties(actor,obj);
return actorRepo.save(obj);
})
.onErrorResume(e->{
return Mono.error(new RuntimeException("Actor object did not found for updation",e));
});
}
}
//Controller class
package com.nt.rest;
import org.springframework.beans.factory.annotation.Autowired; import
org.springframework.http.HttpStatus; import org.springframework.http.ResponseEntity; import
org.springframework.web.bind.annotation.DeleteMapping; import
org.springframework.web.bind.annotation.GetMapping; import
org.springframework.web.bind.annotation.PathVariable; import
org.springframework.web.bind.annotation.PostMapping; import org.springframework.web.bind.annotation.
PutMapping; import org.springframework.web.bind.annotation.RequestBody; import
org.springframework.web.bind.annotation.RequestMapping; import
org.springframework.web.bind.annotation.RestController;
import com.nt.document.ActorInfo;
import com.nt.service.IActorMgmtService;
import reactor.core.publisher.Flux;
import reactor.core.publisher.Mono;
@RestController
@ExceptionHandler(Exception.class)
public ResponseEntity<String> handleAllExceptions(Exception e){
}
return new ResponseEntity<String>("Internal problem:" + e.getMessage(),
HttpStatus.INTERNAL_SERVER_ERROR);
}
@RequestMapping("/actor-api") //global path
public class ActorOperationsController {
@Autowired
private IActorMgmtService service;
@PostMapping("/save")
```

public ResponseEntity<Mono<ActorInfo>> saveActor(@RequestBody ActorInfo info){

//use service

Mono<ActorInfo> mono-service.saveActor(info);

return new ResponseEntity<Mono<ActorInfo>>(mono, HttpStatus.CREATED);

}

@GetMapping("/all")

}

In spring/spring boot Programming, we can use spring webflux module for reactive programming .Using this web flux,we can develop the web applications, Distributed Apps and MicroService Architecture Apps having Reactive programming support

RxJava

Java

Mutiny Vert.x

Java

Akka Streams

Java, Kotlin, Scala Java, Scala

Android, backend services

Quarkus microservices

High-performance, event-driven microservices

Java Flow API

Java

RxJS

JavaScript/TypeScript

Distributed systems Native Java applications Angular, frontend apps

Node.js Streams

JavaScript

Streaming in Node.js

Rx.NET

C#

RxPY

Kotlin Flow

Python Kotlin

RxGo

Go

.NET applications

Data pipelines, async processing Android, Kotlin backend

Reactive programming in Go

=> Different Technologies/frameworks communication b/w

use different techniques to bring asynchronous

Client and Server app

a) Jquery, angular and etc..

b) spring webflux module or spring Reactive module internally depends on Event Loop mechanism to do the same

internally depends on Ajax Engine support to do the same

X What ReactJS is not:

•

It is not a reactive programming framework like RxJS, Recoil, or MobX.

•

It doesn't natively use event streams, observables, or functional reactive patterns.

Then why the name "React"?

The name "React" comes from its ability to "react" to changes in data/state and update the UI accordingly.

This is more of a reactive UI update pattern, not full-fledged reactive programming.

```
public ResponseEntity<Flux<ActorInfo>> findAllActors(){
System.out.println("ActorOperationsController.findAllActors() Thread name::"
+Thread.currentThread().getName());
//use service
Flux<ActorInfo> flux-service.showAllActors();
return new ResponseEntity<Flux<ActorInfo>>(flux, HttpStatus.OK);
@GetMapping("/find/{id}")
public ResponseEntity<Mono<ActorInfo>> findActorById(@PathVariable Integer id){
}
System.out.println("ActorOperationsController.findActorById() Thread name::"
+Thread.currentThread().getName());
//use Service
Mono<ActorInfo> mono-service.showActorById(id);
return new ResponseEntity<Mono<ActorInfo>>(mono, HttpStatus.OK);
@PutMapping("/update")
public ResponseEntity<Mono<ActorInfo>> updateActor(@RequestBody ActorInfo actor){
Mono<ActorInfo> msg=service.updateActor(actor);
//use service
return new ResponseEntity<Mono<ActorInfo>>(msg, HttpStatus.OK);
}
@DeleteMapping("/delete/{id}")
public ResponseEntity<String> deleteActor(@PathVariable Integer id){
//use service
service.removeActorById(id);
```

return new ResponseEntity<String>("Document is deleted", HttpStatus.OK);

}

}

==> Test the Provider App or Reactive API using POSTMAN Tool

Consumer App development using WebClient for Reactive Provider App or Reaactive REST API

=====

BootWebFluxProj01-ReactiveConsumerApp [boot] [dev

=======

✓ src/main/java

#com.nt

> BootWebFluxProj01 Reactive ConsumerAppApp

Model class

10

com.nt.document

==========

› ActorInfo.java

1:

com.nt.runner

1:

package com.nt.document;

>

DeleteDocTestRunner.java

1:

> FindAllDocs TestRunner.java

import org.springframework.data.annotation.Id;

14

>

FindSingleDocTestRunner.java

import org.springframework.data.mongodb.core.mapping.Document;

> Save DocumentTestRunner.java

1!

>

UpdateDocTestRunner.java

11

import lombok.AllArgsConstructor;

src/main/resources

import lombok.Data;

application.properties

import lombok.NoArgsConstructor;

>

src/test/java

F

>

JRE System Library [JavaSE-17]

Boc

@Document

> Maven Dependencies

@Data

20

target/generated-sources/annotations

@AllArgsConstructor

20

>

>

src

> target

HELP.md

target/generated-test-sources/test-annotations

@NoArgsConstructor

20

public class ActorInfo {

AC

@Id

20

private Integer aid;

mvnw

mvnw.cmd

pom.xml

private String aname;

Dc

private String category;

Ac

private Double fee;

}

```
//Runner1
========
package com.nt.runner;
import org.springframework.boot.Command Line Runner;
import org.springframework.stereotype.Component;
import org.springframework.web.reactive.function.client.WebClient;
import com.nt.document.ActorInfo;
import reactor.core.publisher.Mono;
@Component
public class Save DocumentTestRunner implements CommandLineRunner {
}
@Override
public void run(String... args) throws Exception {
//1. Create WebClient Object
WebClient client = WebClient.create("http://localhost:4041");
//2. method, Path(URI), retrieve-type
Mono<ActorInfo> mono = client
.post() //for post mode request
.uri("/actor-api/save") // endpoint url
.body(Mono.just(new ActorInfo(1083, "nagesh", "hyd113",51000.0)), ActorInfo.class) //gives ActorInfo class
obj as JSON data
.retrieve() //calls provider rest api endpoint
.bodyToMono(ActorInfo.class); // converts the recieved JSON response into ActorInfo class obj
mono.subscribe(System.out::println);
//System.exit(0);
}
//Runner2
=========
package com.nt.runner;
import org.springframework.boot.Command LineRunner;
import org.springframework.stereotype.Component; import
org.springframework.web.reactive.function.client.WebClient;
import com.nt.document.ActorInfo;
import reactor.core.publisher.Mono;
@Component
public class FindSingleDocTestRunner implements CommandLineRunner {
}
@Override
```

```
public void run(String... args) throws Exception {
//1. Create WebClient Object
WebClient client = WebClient.create("http://localhost:4041");
//2. method, Path(URI), retrieve-type
Mono<ActorInfo> mono = client
.get() //GET mode_request
.uri("/actor-api/find/1081") //endpojt url 1083 is the param value
.retrieve() //invokes the endpoint
.bodyToMono(ActorInfo.class); // converts the recieved JSON response into ActorInfo class obj
mono.subscribe(System.out::println);
//Runner3
package com.nt.runner;
import org.springframework.boot.Command LineRunner;
import org.springframework.stereotype.Component; import
org.springframework.web.reactive.function.client.WebClient;
import com.nt.document.ActorInfo;
import reactor.core.publisher.Flux;
@Component
public class FindAllDocs TestRunner implements CommandLineRunner {
}
//@Override
public void run(String... args) throws Exception {
//1. Create WebClient Object
WebClient client = WebClient.create("http://localhost:4041");
//invoke api operation
Flux<ActorInfo> flux = client
.get() //GET Mode request
.uri("/actor-api/all") //endpoint url
.retrieve() //invoke the provider API's method
.bodyToFlux(ActorInfo.class); // converts the recived json repsonse into Flux<ActorInfo> object
//3.print (loop)
flux.doOnNext(System.out::println).blockLast();
//Runner4
package com.nt.runner;
import org.springframework.boot.Command Line Runner;
import org.springframework.stereotype.Component; import
org.springframework.web.reactive.function.client.WebClient;
import com.nt.document.ActorInfo;
```

```
import reactor.core.publisher.Mono;
@Component
public class UpdateDocTestRunner implements Command Line Runner {
@Override
public void run(String... args) throws Exception {
//1. Create WebClient Object
WebClient client = WebClient.create("http://localhost:4041");
//2. method,Path(URI), retrieve-type
Mono<ActorInfo> mono =
client.put() //DELETE mode request
.uri("/actor-api/update") //endpoint url
.body(Mono.just(new ActorInfo(1081,"mahesh babu", "hyd345",93000.0)), Actorinfo.class) //gives ActorInfo class obj as JSON data
.retrieve() // invokes Provider API's Endpoint
}
}
.bodyToMono(ActorInfo.class); //converting recieved response to Mono<Void> object
mono.subscribe(System.out::println);
System.out.println("ACtorInfo object is Updated");
//Runner5
==========
package com.nt.runner;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component; import
org.springframework.web.reactive.function.client.WebClient;
import reactor.core.publisher.Mono;
@Component
public class DeleteDocTestRunner implements CommandLineRunner {
}
@Override
public void run(String... args) throws Exception {
//1. Create WebClient Object
WebClient client = WebClient.create("http://localhost:4041");
//2. method, Path(URI), retrieve-type
Mono<String> mono =
client.delete() //DELETE mode request
.uri("/actor-api/delete/1081") //endpoint url
.retrieve() // invokes Provider API's Endpoint
```

```
.bodyToMono(String.class); //converting recieved response to Mono<Void> object
mono.subscribe(System.out::println);
//System.out.println("Deleted!!");
```