THE DEVELOPER'S CONFERENCE

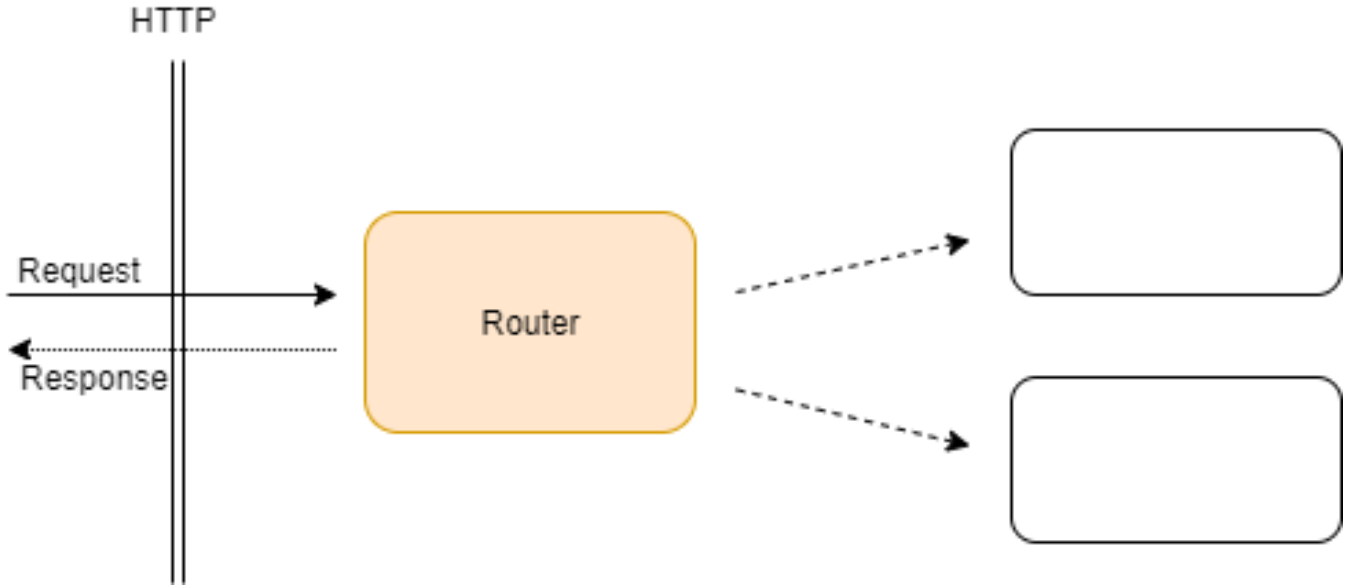
Spring MVC ou Webflux

Escolhendo a melhor stack para sua aplicação

Renan Roggia - Ironi Medina

Breve contexto

THE DEVELOPER'S CONFERENCE



Agenda

THE DEVELOPER'S CONFERENCE

- Escolhendo a stack
  - Linguagem de programação
  - Web framework
- Spring Webflux
  - Diferenças
    - Paradigmas
    - Modelos de concorrências
- O que deu certo
- O que deu errado
- Escolhendo sua stack Spring

THE
DEVELOPER'S
CONFERENCE

Escolhendo a linguagem de programação

Primeira escolha

THE DEVELOPER'S CONFERENCE

- Requerimento: Performance
- Baseline de performance
  - Java + Spring
  - Python + flask
  - Go
  - Javascript + express
- Teste de performance: basic routing

Primeira escolha

THE DEVELOPER'S CONFERENCE

- Requerimento: Performance
- Baseline de performance
  - Java + Spring
  - Python + flask
  - Go
  - Javascript + express
- Teste de performance: basic routing

Primeira escolha

THE DEVELOPER'S CONFERENCE

- Requerimento: Performance
- Baseline de performance
  - Java + Spring
  - Python + flask
  - Go
  - Javascript + express
- Teste de performance: basic routing

Decisão

THE DEVELOPER'S CONFERENCE

- Java + Spring
- Maturidade técnica do time e projeto

THE DEVELOPER'S CONFERENCE

Escolhendo o Web Framework

Segunda escolha

THE DEVELOPER'S CONFERENCE

- Requerimento: Performance e produtividade
- Baseline de performance
  - MVC
  - Webflux
- Teste de performance: routing
  - Com autenticação
  - HTTP vs AMQP

Segunda escolha

THE DEVELOPER'S CONFERENCE

- Requerimento: Performance e produtividade
- Baseline de performance
  - MVC
  - Webflux
- Teste de performance: routing
  - Com autenticação
  - HTTP vs AMQP

Segunda escolha

THE DEVELOPER'S CONFERENCE

- Requerimento: Performance e produtividade
- Baseline de performance
  - MVC
  - Webflux
- Teste de performance: routing
  - Com autenticação
  - HTTP vs AMQP

O que é o Spring Webflux

THE DEVELOPER'S CONFERENCE

- Versão reativa do Spring MVC
  - Novo modelo de concorrência
  - Baseado no projeto Reactor 3
  - Suporte a operações non-blocking
  - Endpoints funcionais
  - Suporte a testes reativos

Spring WebFlux

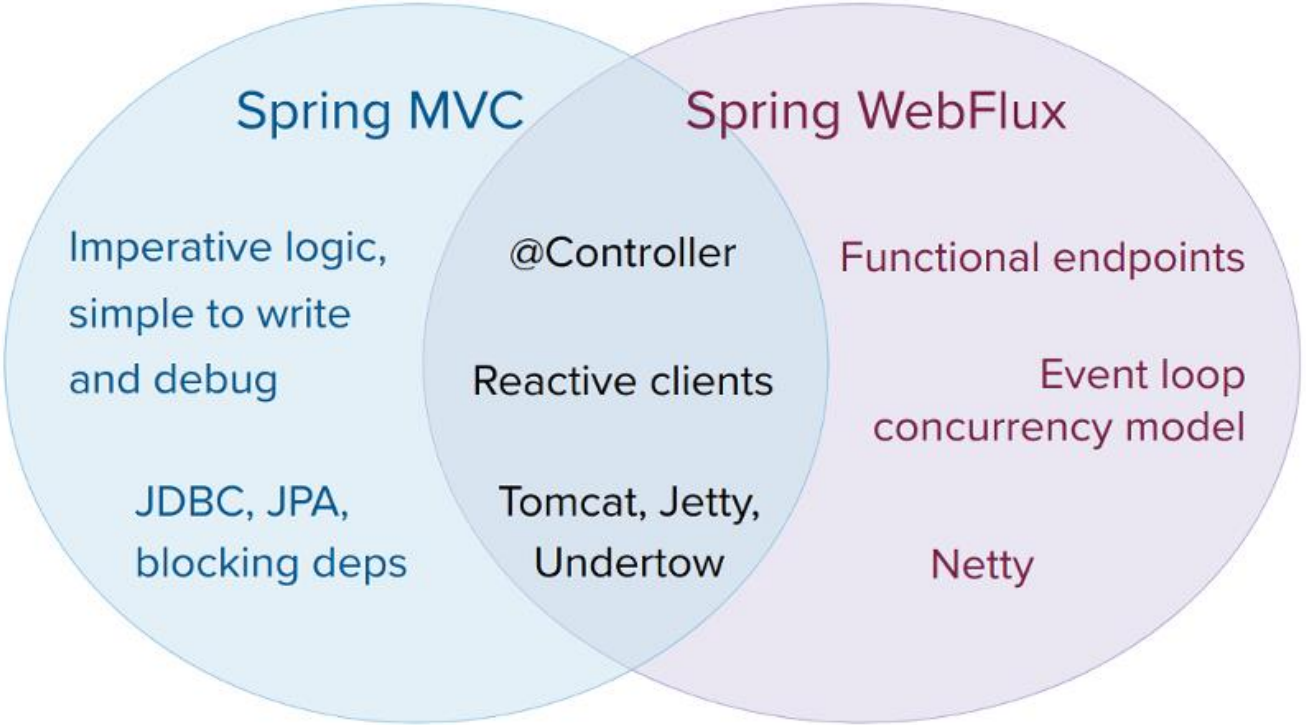
THE DEVELOPER'S CONFERENCE

"For a non-blocking web stack to handle concurrency with a small number of threads and scale with fewer hardware resources."

Source: https://docs.spring.io/spring/docs/current/spring-framework-reference/web-reactive.html

O que é o Spring Webflux

THE DEVELOPER'S CONFERENCE



Diferenças Técnicas

THE DEVELOPER'S CONFERENCE

MVC

- Servlet API
- Servlet Container
- Modelo de concorrência: One Thread per request
  - Thread pool grande
- Paradigma: Imperativo e Funcional

Webflux

- Reactive Streams
- Netty, Servlet Container (+3.1)
- Modelo de concorrência: Event loop
  - Thread per core
  - Worker thread para I/O
- Paradigma: Reativo

Diferenças Técnicas

THE DEVELOPER'S CONFERENCE

MVC

- Servlet API
- Servlet Container
- Modelo de concorrência: One Thread per request
  - Thread pool grande
- Paradigma: Imperativo e Funcional

Webflux

- Reactive Streams
- Netty, Servlet Container (+3.1)
- Modelo de concorrência: Event loop
  - Thread per core
  - Worker thread para I/O
- Paradigma: Reativo

Paradigmas

THE DEVELOPER'S CONFERENCE

- Forma como pensamos e vemos o mundo em termos de programação

- Várias formas de fazer; diferentes paradigmas
  - dada uma lista de jogadores, filtrar apenas os que possuem mais de 35 anos e jogam no Brasil, e transformar em passível de aposentadoria

```java
/**
 * Imperative = HOW
 */
private static void playersToRetireImperative() {
    for (int i = 0; i < players.size(); i++) {
        Player player = players.get(i);

        if (player.getAge() > RETIREMENT_AGE) {
            if (CountryCode.BR.equals(player.getCountry())) {
                PlayerToRetire toRetire = new PlayerToRetire(player);
                toRetire.setRetired(true);

                System.out.println(toRetire);
            }
        }
    }
}
```

Paradigmas - Imperativo

THE DEVELOPER'S CONFERENCE

- mais simples e mais utilizado
- fácil compreensão e aprendizagem
- passo a passo do que queremos fazer
- variáveis para guardar estado
- for/while para controlar fluxo
- foca em COMO o processo deve ser feito

```java
/**
 * Functional = WHAT
 */
private static void playersToRetireFunctional() {
    players.stream()
        .filter(player -> player.getAge() > RETIREMENT_AGE)
        .filter(player -> CountryCode.BR.equals(player.getCountry()))
        .map(player -> {
            PlayerToRetire toRetire = new PlayerToRetire(player);
            toRetire.setRetired(true);
            return toRetire;
        })
        .forEach(System.out::println);
}
```

Paradigmas - Funcional

THE DEVELOPER'S CONFERENCE

- não guarda estado
- não existe mutação de variáveis
- resultado da função depende somente do input
- recursividade para iterações
- define O QUE deve ser feito sem se preocupar como

```java
/**
 * Reactive = WHEN + WHAT
 */
private static void playersToRetireReactive() {
    Flux<PlayerToRetire> toRetirePublisher = playersReactive
        .filter(player -> player.getAge() > RETIREMENT_AGE)
        .filter(player -> CountryCode.BR.equals(player.getCountry()))
        .map(player -> {
            PlayerToRetire toRetire = new PlayerToRetire(player);
            toRetire.setRetired(true);
            return toRetire;
        });

    toRetirePublisher.subscribe(System.out::println);
}
```

Paradigmas - Reativo

THE DEVELOPER'S CONFERENCE

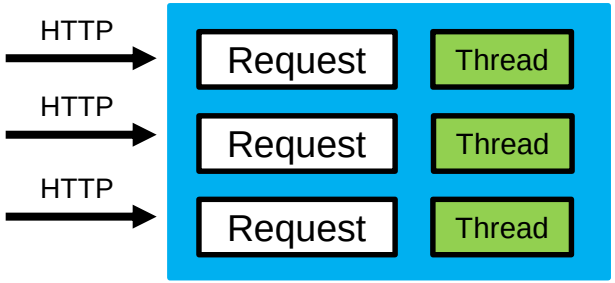
- também conhecido como funcional abstrato
- mais complexo dos três
- fluxo de dados assíncronos
- eventos que podem ocorrer em diferentes momentos
- aplica funções (funcional) quando o dado está disponível

Paradigmas

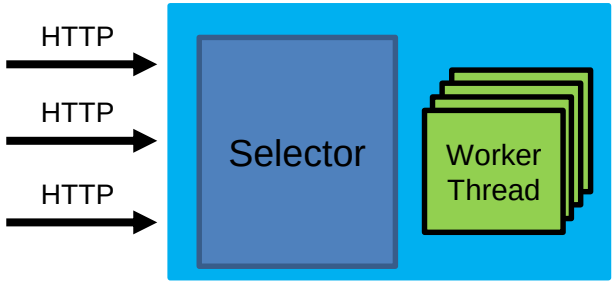
THE
DEVELOPER'S
CONFERENCE

Modelos de Concorrência

THE DEVELOPER'S CONFERENCE

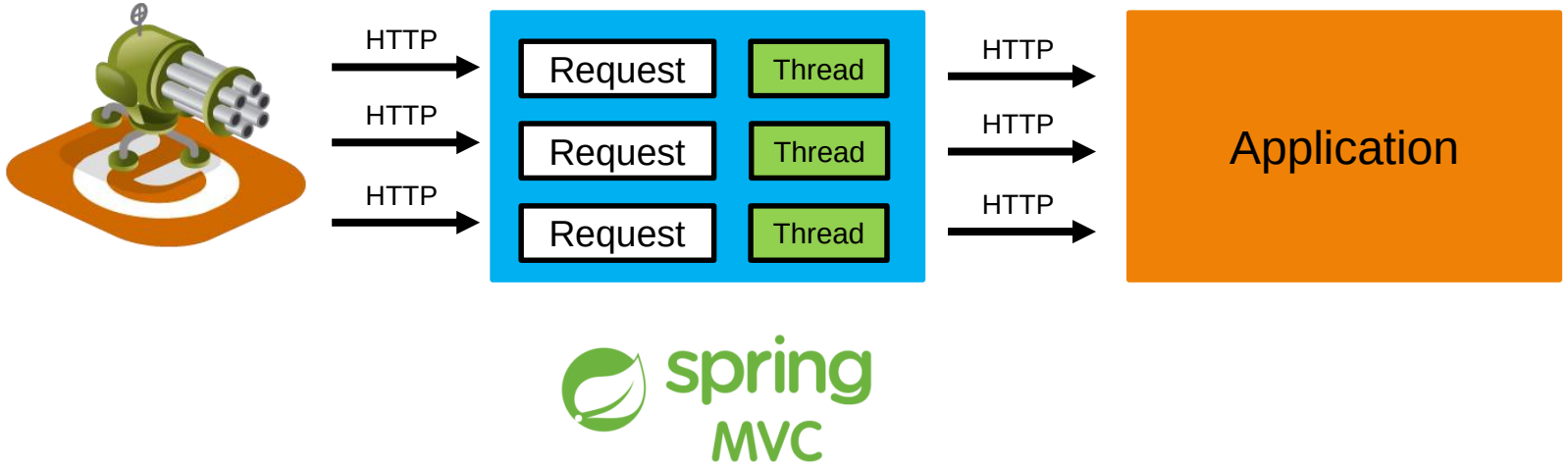


spring MVC



spring WEBFLUX

One thread per request model



```java
@RestController
public class RoutingController {

    private static final String ENGINE_URL = "http://localhost:8081/route";

    @GetMapping(path = "route", produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
    public ResponseEntity<String> route(@RequestParam String delay) {

        String uri = UriComponentsBuilder.fromHttpUrl(ENGINE_URL).queryParam("delay", delay).toUriString();

        try {
            String response = new RestTemplate().getForObject(uri, String.class);
            return ResponseEntity.ok().body(response);
        } catch (HttpClientErrorException.BadRequest | HttpServerErrorException.InternalServerError e) {
            return ResponseEntity.badRequest().body(e.getResponseBodyAsString());
        }
    }
}
```

```java
@RestController
public class RoutingController {

    private static final String ENGINE_URL = "http://localhost:8081/route";

    @GetMapping(path = "route", produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
    public ResponseEntity<String> route(@RequestParam String delay) {

        String uri = UriComponentsBuilder.fromHttpUrl(ENGINE_URL).queryParam("delay", delay).toUriString();

        try {
            String response = new RestTemplate().getForObject(uri, String.class);
            return ResponseEntity.ok().body(response);
        } catch (HttpClientErrorException.BadRequest | HttpServerErrorException.InternalServerError e) {
            return ResponseEntity.badRequest().body(e.getResponseBodyAsString());
        }
    }
}
```

```java
@RestController
public class RoutingController {

    private static final String ENGINE_URL = "http://localhost:8081/route";

    @GetMapping(path = "route", produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
    public ResponseEntity<String> route(@RequestParam String delay) {

        String uri = UriComponentsBuilder.fromHttpUrl(ENGINE_URL).queryParam("delay", delay).toUriString();

        try {
            String response = new RestTemplate().getForObject(uri, String.class);
            return ResponseEntity.ok().body(response);
        } catch (HttpClientErrorException.BadRequest | HttpServerErrorException.InternalServerError e) {
            return ResponseEntity.badRequest().body(e.getResponseBodyAsString());
        }
    }
}
```

```java
@RestController
public class RoutingController {

    private static final String ENGINE_URL = "http://localhost:8081/route";

    @GetMapping(path = "route", produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
    public ResponseEntity<String> route(@RequestParam String delay) {

        String uri = UriComponentsBuilder.fromHttpUrl(ENGINE_URL).queryParam("delay", delay).toUriString();

        try {
            String response = new RestTemplate().getForObject(uri, String.class);
            return ResponseEntity.ok().body(response);
        } catch (HttpClientErrorException.BadRequest | HttpServerErrorException.InternalServerError e) {
            return ResponseEntity.badRequest().body(e.getResponseBodyAsString());
        }
    }
}
```


Live threads: 119
Daemon threads: 115
Running
Sleeping
Wait
Park
Monitor
Timeline
View: Live threads
Name
17:20:40
17:20:45
17:20:50
17:20:55
17:21:00
17:21:05
Running
Total
http-nio-8080-exec-112 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-113 93.055 ms (85,3%) 109.034 ms
http-nio-8080-exec-114 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-115 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-116 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-117 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-118 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-119 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-120 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-121 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-122 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-123 92.015 ms (84,4%) 109.034 ms
http-nio-8080-exec-124 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-125 92.016 ms (84,4%) 109.034 ms
http-nio-8080-exec-126 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-127 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-128 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-129 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-13 485.439 ms (4,5%) 10.794.783 ms
http-nio-8080-exec-130 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-131 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-132 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-133 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-134 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-135 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-136 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-137 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-138 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-139 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-140 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-141 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-142 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-143 94.040 ms (86,2%) 109.034 ms
http-nio-8080-exec-144 94.040 ms (86,2%) 109.034 ms

Síncrono e bloqueante

THE DEVELOPER'S CONFERENCE

```
"http-nio-8080-exec-102" - Thread t@218
   java.lang.Thread.State: RUNNABLE
        at java.net.SocketInputStream.socketRead0(Native Method)
        at java.net.SocketInputStream.socketRead(SocketInputStream.java:116)
        at java.net.SocketInputStream.read(SocketInputStream.java:171)
```

THE DEVELOPER'S CONFERENCE

"Waiting within the servlet is an inefficient operation as it is a blocking operation that consumes a thread and other limited resources."

Java Servlet Specification 3.1

THE
DEVELOPER'S
CONFERENCE

Async e bloqueante

Feio mas arrumadinho

```java
@RestController
public class RoutingController {

    private static final String ENGINE_URL = "http://localhost:8081";

    @GetMapping(path = "anotherRoute", produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
    public DeferredResult<ResponseEntity<String>> anotherRoute(@RequestParam String delay) {
        DeferredResult<ResponseEntity<String>> response = new DeferredResult<>();

        ForkJoinPool.commonPool().submit(() -> {
            String uri = UriComponentsBuilder.fromHttpUrl(ENGINE_URL).queryParam("delay", delay).toUriString();

            try {
                response.setResult(ResponseEntity.ok().body(new RestTemplate().getForObject(uri, String.class)));
            } catch (HttpClientErrorException.BadRequest | HttpServerErrorException.InternalServerError e) {
                response.setResult(ResponseEntity.badRequest().body(e.getResponseBodyAsString()));
            }

        });

        return response;
    }
}
```

```java
@RestController
public class RoutingController {

    private static final String ENGINE_URL = "http://localhost:8081";

    @GetMapping(path = "anotherRoute", produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
    public DeferredResult<ResponseEntity<String>> anotherRoute(@RequestParam String delay) {
        DeferredResult<ResponseEntity<String>> response = new DeferredResult<>();

        ForkJoinPool.commonPool().submit(() -> {
            String uri = UriComponentsBuilder.fromHttpUrl(ENGINE_URL).queryParam("delay", delay).toUriString();

            try {
                response.setResult(ResponseEntity.ok().body(new RestTemplate().getForObject(uri, String.class)));
            } catch (HttpClientErrorException.BadRequest | HttpServerErrorException.InternalServerError e) {
                response.setResult(ResponseEntity.badRequest().body(e.getResponseBodyAsString()));
            }

        });

        return response;
    }
}
```

```java
@RestController
public class RoutingController {

    private static final String ENGINE_URL = "http://localhost:8081";

    @GetMapping(path = "anotherRoute", produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
    public DeferredResult<ResponseEntity<String>> anotherRoute(@RequestParam String delay) {
        DeferredResult<ResponseEntity<String>> response = new DeferredResult<>();

        ForkJoinPool.commonPool().submit(() -> {
            String uri = UriComponentsBuilder.fromHttpUrl(ENGINE_URL).queryParam("delay", delay).toUriString();

            try {
                response.setResult(ResponseEntity.ok().body(new RestTemplate().getForObject(uri, String.class)));
            } catch (HttpClientErrorException.BadRequest | HttpServerErrorException.InternalServerError e) {
                response.setResult(ResponseEntity.badRequest().body(e.getResponseBodyAsString()));
            }

        });

        return response;
    }
}
```

```java
@RestController
public class RoutingController {

    private static final String ENGINE_URL = "http://localhost:8081";

    @GetMapping(path = "anotherRoute", produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
    public DeferredResult<ResponseEntity<String>> anotherRoute(@RequestParam String delay) {
        DeferredResult<ResponseEntity<String>> response = new DeferredResult<>();

        ForkJoinPool.commonPool().submit(() -> {
            String uri = UriComponentsBuilder.fromHttpUrl(ENGINE_URL).queryParam("delay", delay).toUriString();

            try {
                response.setResult(ResponseEntity.ok().body(new RestTemplate().getForObject(uri, String.class)));
            } catch (HttpClientErrorException.BadRequest | HttpServerErrorException.InternalServerError e) {
                response.setResult(ResponseEntity.badRequest().body(e.getResponseBodyAsString()));
            }

        });

        return response;
    }
}
```

```java
@RestController
public class RoutingController {

    private static final String ENGINE_URL = "http://localhost:8081";

    @GetMapping(path = "anotherRoute", produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
    public DeferredResult<ResponseEntity<String>> anotherRoute(@RequestParam String delay) {
        DeferredResult<ResponseEntity<String>> response = new DeferredResult<>();

        ForkJoinPool.commonPool().submit(() -> {
            String uri = UriComponentsBuilder.fromHttpUrl(ENGINE_URL).queryParam("delay", delay).toUriString();

            try {
                response.setResult(ResponseEntity.ok().body(new RestTemplate().getForObject(uri, String.class)));
            } catch (HttpClientErrorException.BadRequest | HttpServerErrorException.InternalServerError e) {
                response.setResult(ResponseEntity.badRequest().body(e.getResponseBodyAsString()));
            }

        });

        return response;
    }
}
```

THE DEVELOPER'S CONFERENCE

Async e Não bloqueante

Tekpix

```java
@RestController
public class RoutingController {

    private static final String ENGINE_URL = "http://localhost:8081";
    private static WebClient webClient = WebClient.create(ENGINE_URL);

    @GetMapping(path = "route", produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
    public Mono<String> route(@RequestParam String delay) {

        return webClient.get().uri("/route?delay=" + delay)
                .header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE).retrieve()
                .onStatus(HttpStatus::is4xxClientError, e -> Mono.error(new RuntimeException("e")))
                .onStatus(HttpStatus::is5xxServerError, e -> Mono.error(new RuntimeException("e")))
                .bodyToMono(String.class);

    }
}
```

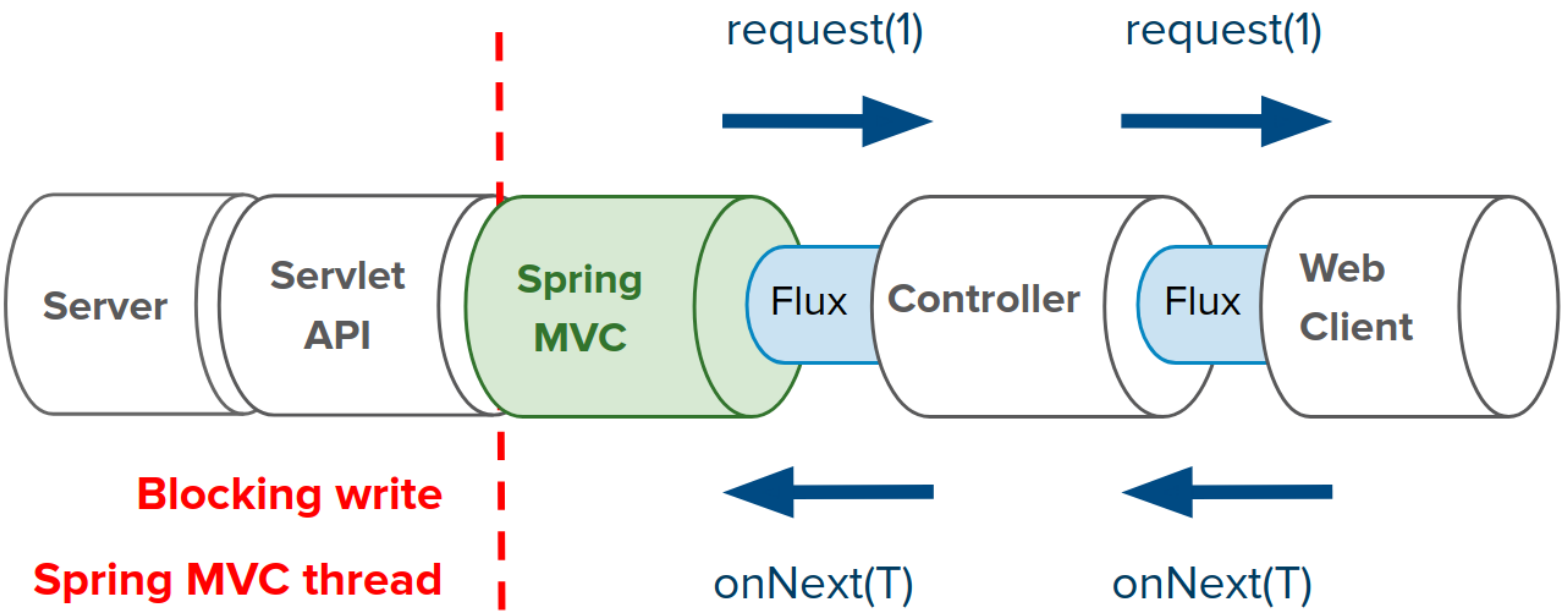
```java
@RestController
public class RoutingController {

    private static final String ENGINE_URL = "http://localhost:8081";
    private static WebClient webClient = WebClient.create(ENGINE_URL);

    @GetMapping(path = "route", produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
    public Mono<String> route(@RequestParam String delay) {

        return webClient.get().uri("/route?delay=" + delay)
                .header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE).retrieve()
                .onStatus(HttpStatus::is4xxClientError, e -> Mono.error(new RuntimeException("e")))
                .onStatus(HttpStatus::is5xxServerError, e -> Mono.error(new RuntimeException("e")))
                .bodyToMono(String.class);

    }
}
```

```java
@RestController
public class RoutingController {

    private static final String ENGINE_URL = "http://localhost:8081";
    private static WebClient webClient = WebClient.create(ENGINE_URL);

    @GetMapping(path = "route", produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
    public Mono<String> route(@RequestParam String delay) {

        return webClient.get().uri("/route?delay=" + delay)
                .header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE).retrieve()
                .onStatus(HttpStatus::is4xxClientError, e -> Mono.error(new RuntimeException("e")))
                .onStatus(HttpStatus::is5xxServerError, e -> Mono.error(new RuntimeException("e")))
                .bodyToMono(String.class);

    }
}
```

```java
@RestController
public class RoutingController {

    private static final String ENGINE_URL = "http://localhost:8081";
    private static WebClient webClient = WebClient.create(ENGINE_URL);

    @GetMapping(path = "route", produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
    public Mono<String> route(@RequestParam String delay) {

        return webClient.get().uri("/route?delay=" + delay)
                .header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE).retrieve()
                .onStatus(HttpStatus::is4xxClientError, e -> Mono.error(new RuntimeException("e")))
                .onStatus(HttpStatus::is5xxServerError, e -> Mono.error(new RuntimeException("e")))
                .bodyToMono(String.class);

    }
}
```

```java
@RestController
public class RoutingController {

    private static final String ENGINE_URL = "http://localhost:8081";
    private static WebClient webClient = WebClient.create(ENGINE_URL);

    @GetMapping(path = "route", produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
    public Mono<String> route(@RequestParam String delay) {

        return webClient.get().uri("/route?delay=" + delay)
                .header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE).retrieve()
                .onStatus(HttpStatus::is4xxClientError, e -> Mono.error(new RuntimeException("e")))
                .onStatus(HttpStatus::is5xxServerError, e -> Mono.error(new RuntimeException("e")))
                .bodyToMono(String.class);

    }
}
```


org.codehaus.plexus.classworlds.launcher.Launcher (pid 20768)
Threads
Live threads: 120
Daemon threads: 116
Running
Sleeping
Wait
Park
Monitor
Timeline
View: Live threads
Name
19:05:20
19:05:25
19:05:30
19:05:35
19:05:40
19:05:45
Running
Total
http-nio-8082-exec-10 0 ms (0%) 350.408 ms
http-nio-8082-exec-11 0 ms (0%) 187.844 ms
http-nio-8082-exec-12 1.018 ms (0,5%) 187.844 ms
http-nio-8082-exec-13 0 ms (0%) 187.844 ms
http-nio-8082-exec-14 1.017 ms (0,5%) 187.844 ms
http-nio-8082-exec-15 0 ms (0%) 187.844 ms
http-nio-8082-exec-16 0 ms (0%) 187.844 ms
http-nio-8082-exec-17 954 ms (0,5%) 187.844 ms
http-nio-8082-exec-18 0 ms (0%) 187.844 ms
http-nio-8082-exec-19 0 ms (0%) 187.844 ms
http-nio-8082-exec-2 1.000 ms (0,3%) 350.408 ms
http-nio-8082-exec-20 0 ms (0%) 187.844 ms
http-nio-8082-exec-21 1.044 ms (0,6%) 187.844 ms
http-nio-8082-exec-22 1.004 ms (0,5%) 187.844 ms
http-nio-8082-exec-23 0 ms (0%) 187.844 ms
http-nio-8082-exec-24 0 ms (0%) 187.844 ms
http-nio-8082-exec-25 0 ms (0%) 187.844 ms
http-nio-8082-exec-26 1.017 ms (0,5%) 187.844 ms
http-nio-8082-exec-27 0 ms (0%) 187.844 ms
http-nio-8082-exec-28 0 ms (0%) 187.844 ms
http-nio-8082-exec-29 0 ms (0%) 187.844 ms
http-nio-8082-exec-3 0 ms (0%) 350.408 ms
http-nio-8082-exec-30 0 ms (0%) 187.844 ms
http-nio-8082-exec-31 0 ms (0%) 187.844 ms
http-nio-8082-exec-32 0 ms (0%) 187.844 ms
http-nio-8082-exec-33 0 ms (0%) 187.844 ms
http-nio-8082-exec-34 0 ms (0%) 187.844 ms
http-nio-8082-exec-35 0 ms (0%) 187.844 ms
http-nio-8082-exec-36 0 ms (0%) 187.844 ms
http-nio-8082-exec-37 0 ms (0%) 187.844 ms
http-nio-8082-exec-38 0 ms (0%) 187.844 ms
http-nio-8082-exec-39 0 ms (0%) 187.844 ms
http-nio-8082-exec-4 0 ms (0%) 350.408 ms
http-nio-8082-exec-40 0 ms (0%) 187.844 ms
http-nio-8082-exec-41 0 ms (0%) 187.844 ms
Running
Sleeping
Wait
Park
Monitor



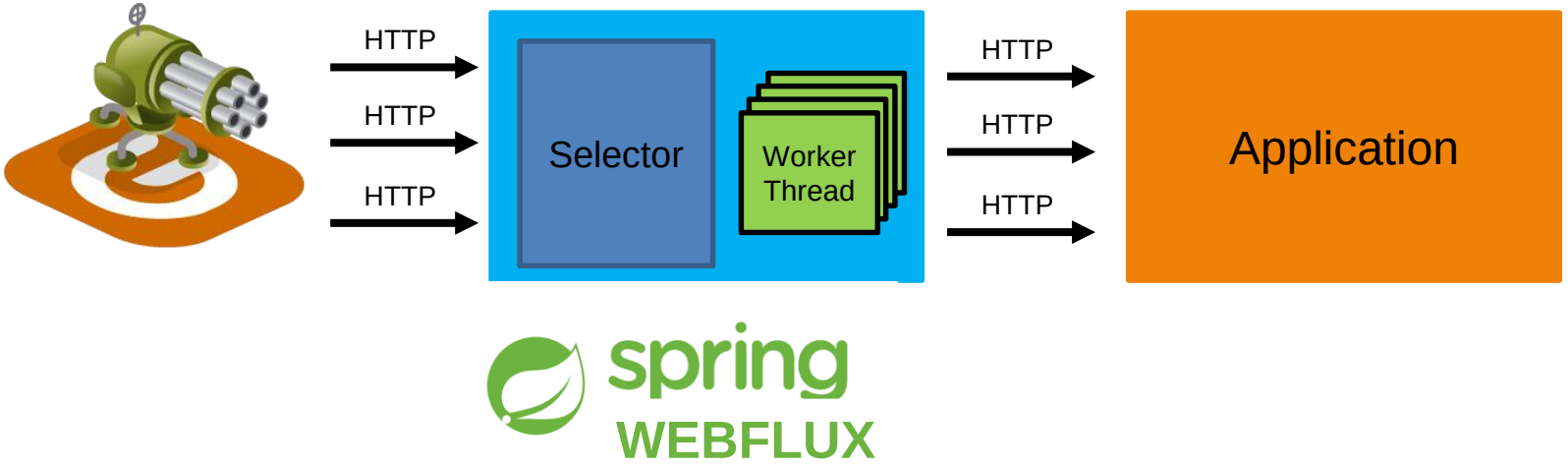
Source: https://www.infoq.com/articles/Servlet-and-Reactive-Stacks-Spring-Framework-5/

THE
DEVELOPER'S
CONFERENCE

Alternativas ao Servlet API?

Spring WebFlux

THE DEVELOPER'S CONFERENCE



```java
@RestController
public class RoutingController {

    private static final String ENGINE_URL = "http://localhost:8081";
    private static WebClient webClient = WebClient.create(ENGINE_URL);

    @GetMapping(path = "route", produces = MediaType.APPLICATION_JSON_UTF8_VALUE)
    public Mono<String> route(@RequestParam String delay) {

        return webClient.get().uri("/route?delay=" + delay)
                .header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE).retrieve()
                .onStatus(HttpStatus::is4xxClientError, e -> Mono.error(new RuntimeException("e")))
                .onStatus(HttpStatus::is5xxServerError, e -> Mono.error(new RuntimeException("e")))
                .bodyToMono(String.class);

    }
}
```

```java
@Configuration
public class RoutingHandler {

    private static WebClient webClient = WebClient.create("http://localhost:8081");

    @Bean
    public RouterFunction<?> routes() {
        return RouterFunctions.route().GET("/route", request -> {
            Optional<String> delay = request.queryParam("delay");

            return webClient.get().uri("/route?delay=" + delay.get())
                    .header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE).retrieve()
                    .bodyToMono(String.class).flatMap(body -> ServerResponse.ok().syncBody(body));

        }).build();
    }
}
```

```java
@Configuration
public class RoutingHandler {

    private static WebClient webClient = WebClient.create("http://localhost:8081");

    @Bean
    public RouterFunction<?> routes() {
        return RouterFunctions.route().GET("/route", request -> {
            Optional<String> delay = request.queryParam("delay");

            return webClient.get().uri("/route?delay=" + delay.get())
                    .header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE).retrieve()
                    .bodyToMono(String.class).flatMap(body -> ServerResponse.ok().syncBody(body));

        }).build();
    }
}
```

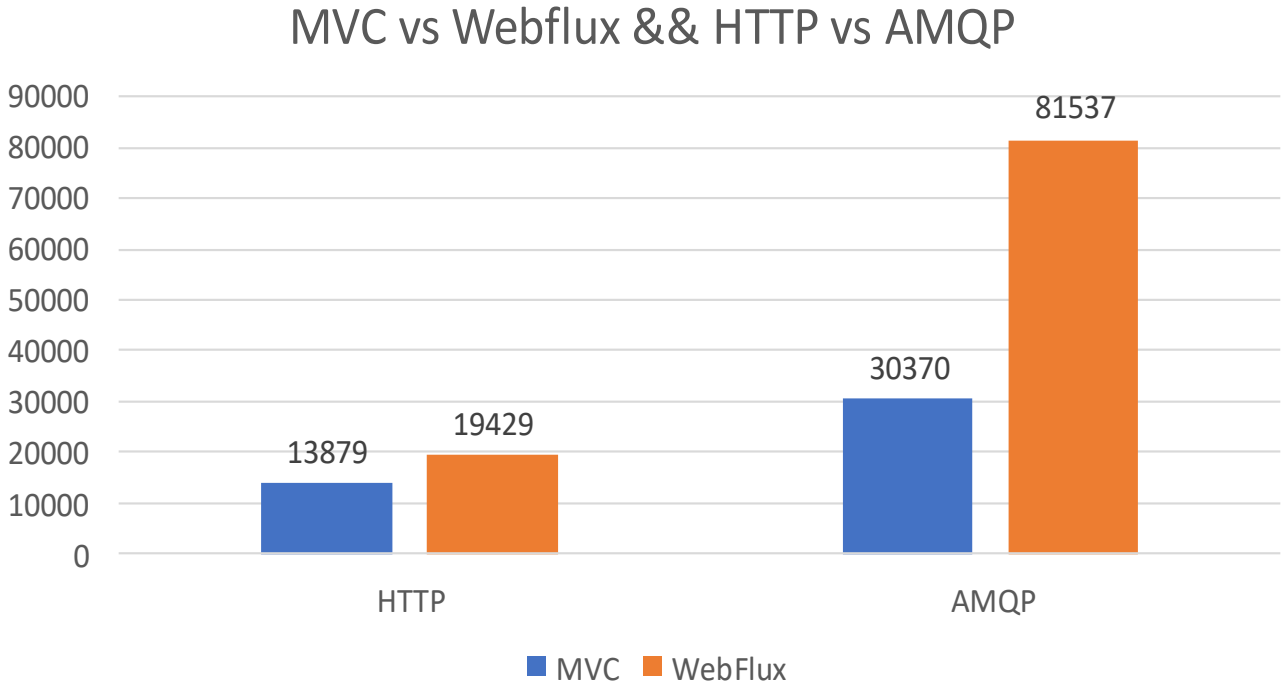
```java
@Configuration
public class RoutingHandler {

    private static WebClient webClient = WebClient.create("http://localhost:8081");

    @Bean
    public RouterFunction<?> routes() {
        return RouterFunctions.route().GET("/route", request -> {
            Optional<String> delay = request.queryParam("delay");

            return webClient.get().uri("/route?delay=" + delay.get())
                    .header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE).retrieve()
                    .bodyToMono(String.class).flatMap(body -> ServerResponse.ok().syncBody(body));

        }).build();
    }
}
```

```java
@Configuration
public class RoutingHandler {

    private static WebClient webClient = WebClient.create("http://localhost:8081");

    @Bean
    public RouterFunction<?> routes() {
        return RouterFunctions.route().GET("/route", request -> {
            Optional<String> delay = request.queryParam("delay");

            return webClient.get().uri("/route?delay=" + delay.get())
                    .header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE).retrieve()
                    .bodyToMono(String.class).flatMap(body -> ServerResponse.ok().syncBody(body));

        }).build();
    }
}
```

```java
@Configuration
public class RoutingHandler {

    private static WebClient webClient = WebClient.create("http://localhost:8081");

    @Bean
    public RouterFunction<?> routes() {
        return RouterFunctions.route().GET("/route", request -> {
            Optional<String> delay = request.queryParam("delay");

            return webClient.get().uri("/route?delay=" + delay.get())
                    .header(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE).retrieve()
                    .bodyToMono(String.class).flatMap(body -> ServerResponse.ok().syncBody(body));

        }).build();
    }
}
```


org.codehaus.plexus.clas
Running
Sleeping
Wait
Park
Monitor
Threads
Threads visualization
Threads inspector
Live threads: 15
Daemon threads: 13
Thread Dump
Timeline
View: All threads
Name
20:12:10
20:12:15
20:12:20
20:12:25
20:12:30
20:12:35
Running
Total
RMI TCP Connection(2)-169.254.5 114.170 ms (100%) 114.170 ms
JMX server connection timeout 24 0 ms (0%) 114.170 ms
RMI Scheduler(0) 0 ms (0%) 114.170 ms
RMI TCP Connection(1)-169.254.5 0 ms (0%) 114.170 ms
RMI TCP Accept-0 114.170 ms (100%) 114.170 ms
reactor-http-nio-4 114.170 ms (100%) 114.170 ms
reactor-http-nio-3 114.170 ms (100%) 114.170 ms
reactor-http-nio-2 114.170 ms (100%) 114.170 ms
server 0 ms (0%) 114.170 ms
reactor-http-nio-1 114.170 ms (100%) 114.170 ms
Attach Listener 114.170 ms (100%) 114.170 ms
Signal Dispatcher 114.170 ms (100%) 114.170 ms
Finalizer 0 ms (0%) 114.170 ms
Reference Handler 0 ms (0%) 114.170 ms
main 0 ms (0%) 114.170 ms
RMI TCP Connection(3)-169.254.5 43.025 ms (41,7%) 103.160 ms

Modelo de concorrência e performance



Melhor utilização de recurso

THE DEVELOPER'S CONFERENCE

- Throughput
- Memória
- Threads

THE
DEVELOPER'S
CONFERENCE

“Se o palestrante não mostra o lado ruim, desconfie”

O que deu errado

THE DEVELOPER'S CONFERENCE

- Precisa do *maldito* subscribe()
- Dependências bloqueantes
  - Autenticação
  - MDC
- Operadores
- Complexidade
- Debugging
- Encadeamento

O que deu errado

THE DEVELOPER'S CONFERENCE

- Precisa do *maldito* subscribe()
- Dependências bloqueantes
  - Autenticação
  - MDC
- Operadores
- Complexidade
- Debugging
- Encadeamento

O que deu errado

THE DEVELOPER'S CONFERENCE

- Precisa do *maldito* subscribe()
- Dependências bloqueantes
  - Autenticação
  - MDC
- Operadores
- Complexidade
- Debugging
- Encadeamento

O que deu errado

THE DEVELOPER'S CONFERENCE

- Precisa do *maldito* subscribe()
- Dependências bloqueantes
  - Autenticação
  - MDC
- Operadores
- Complexidade
- Debugging
- Encadeamento

O que deu errado

THE DEVELOPER'S CONFERENCE

- Precisa do *maldito* subscribe()
- Dependências bloqueantes
  - Autenticação
  - MDC
- Operadores
- Complexidade
- Debugging
- Encadeamento

O que deu errado

THE DEVELOPER'S CONFERENCE

- Precisa do *maldito* subscribe()
- Dependências bloqueantes
  - Autenticação
  - MDC
- Operadores
- Complexidade
- Debugging
- Encadeamento

O que funcionou pra nós

THE DEVELOPER'S CONFERENCE

- Planejar tempo para aprendizado
- Reforçar *Generics* e *java.util.function*
- Testar diferentes operadores
- reactor/lite-rx-api-hands-on
- **Pair programming e refactor**

O que funcionou pra nós

THE DEVELOPER'S CONFERENCE

- Planejar tempo para aprendizado
- Reforçar *Generics* e *java.util.function*
- Testar diferentes operadores
- reactor/lite-rx-api-hands-on
- **Pair programming e refactor**

O que funcionou pra nós

THE DEVELOPER'S CONFERENCE

- Planejar tempo para aprendizado
- Reforçar *Generics* e *java.util.function*
- Testar diferentes operadores
- reactor/lite-rx-api-hands-on
- **Pair programming e refactor**

O que funcionou pra nós

THE DEVELOPER'S CONFERENCE

- Planejar tempo para aprendizado
- Reforçar *Generics* e *java.util.function*
- Testar diferentes operadores
- reactor/lite-rx-api-hands-on
- **Pair programming e refactor**

O que funcionou pra nós

THE DEVELOPER'S CONFERENCE

- Planejar tempo para aprendizado
- Reforçar *Generics* e *java.util.function*
- Testar diferentes operadores
- reactor/lite-rx-api-hands-on
- **Pair programming e refactor**

THE DEVELOPER'S CONFERENCE

Escolhendo sua stack Spring

Concluindo

THE DEVELOPER'S CONFERENCE

- Linguagem de programação
  - Maturidade
- Web framework
  - Diferenças
    - Paradigmas
    - Modelo de concorrências
- Nossas experiências
- Dicas
- Escolhendo sua stack Spring

THANK YOU

Renan Roggia

Software Engineer @ SAP

/RRoggia

/in/renanroggia

renanzr@gmail.com

Ironi Medina

Software Engineer @ SAP

/ironijunior

/in/ironi-junior-medina

ironimedina@gmail.com

THE DEVELOPER'S CONFERENCE

THE DEVELOPER'S CONFERENCE