Going Reactive with Spring 5

JavaSkop'18

Who am I?

Java Technical Lead at Seavus

17 years in the industry

Spring Certified Professional

You can find me at:

- *drazen.nikolic@seavus.com*
- *@drazenis*
- *programminghints.com*

Changing Requirements (then and now)

	10 years ago	*Now*
Server nodes	**10’s**	**1000’s**
Response times	**seconds**	**milliseconds**
Maintenance downtimes	**hours**	**none**
Data volume	**GBs**	**TBs → PBs**

Solution?

EASY: Just spin up more threads!

Reactive Programming

Event-driven systems

Moves imperative logic to:

- asynchronous
- non-blocking
- functional-style code

Allows stable, scalable access to external systems

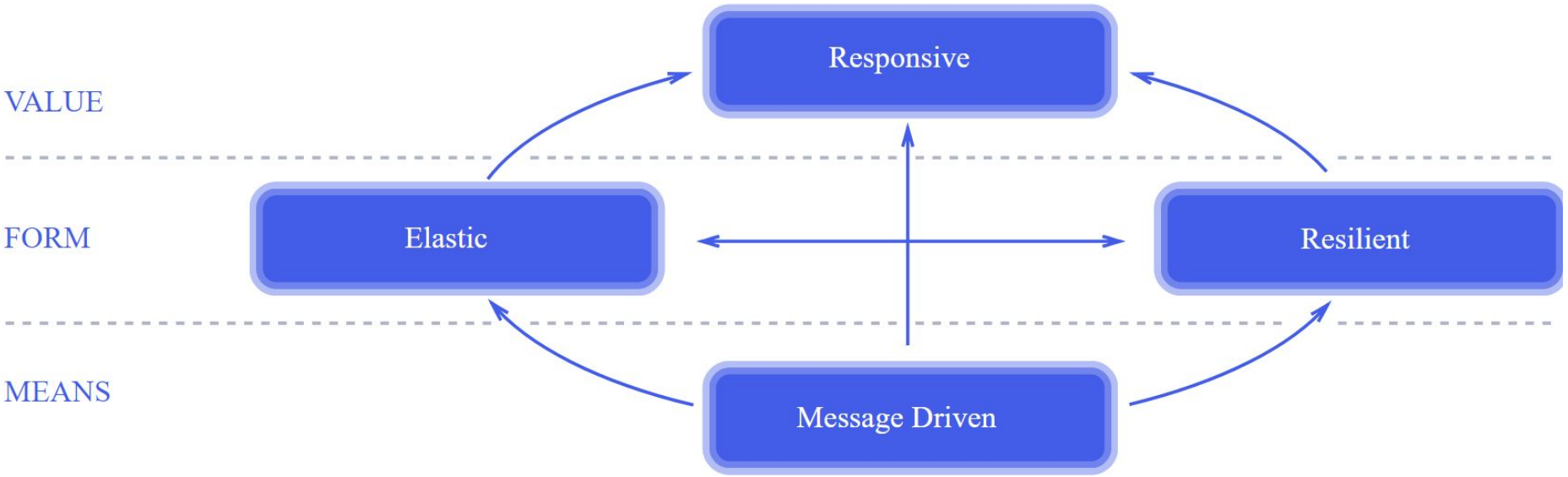
Example use-cases:

Monitoring stock prices
Streaming videos
Spreadsheets
Fraud detection

When to Use Reactive?

- Handling networking issues, like latency or failures
- Scalability concerns
- Clients getting overwhelmed by the sent messages (handling backpressure)
- Highly concurrent operations

Reactive Manifesto



www.reactivemanifesto.org

Reactive Streams Specification

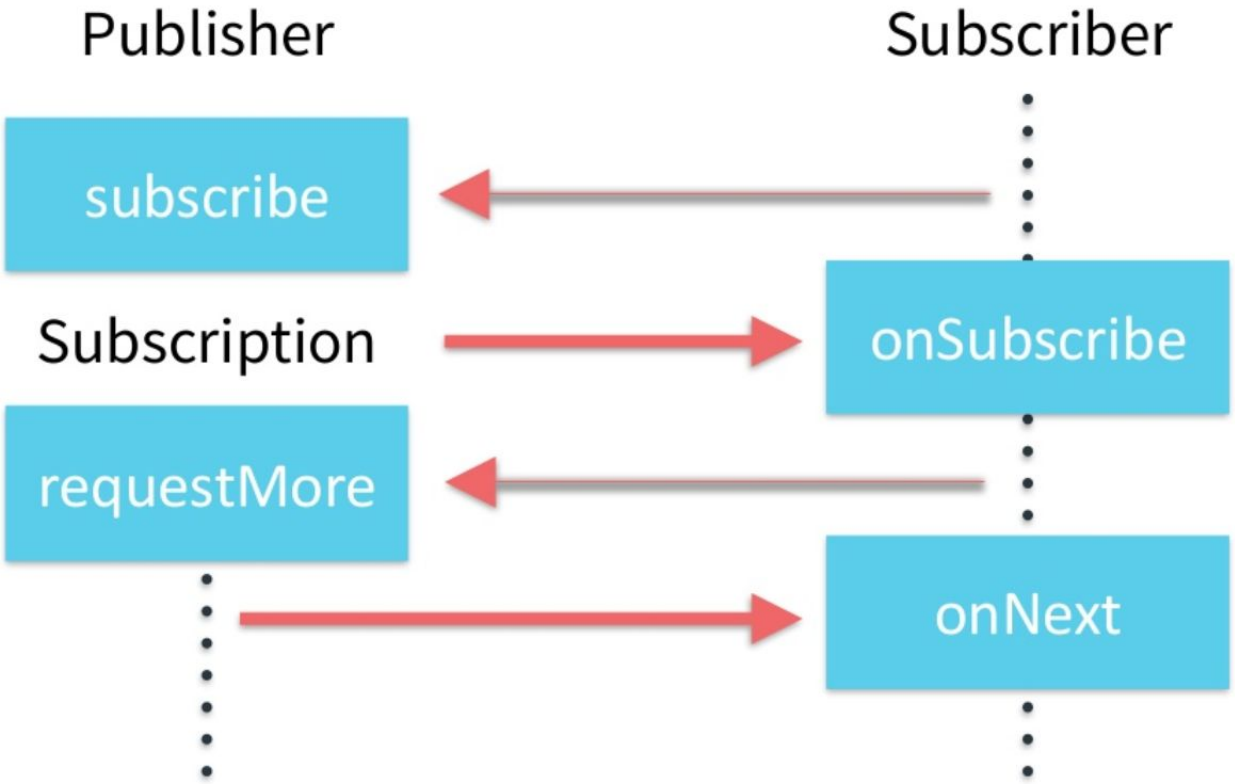
```
public interface Publisher<T> {
    public void subscribe(Subscriber<? super T> s);
}

public interface Subscriber<T> {
    public void onSubscribe(Subscription s);
    public void onNext(T t);
    public void onError(Throwable t);
    public void onComplete();
}

public interface Subscription {
    public void request(long n);
    public void cancel();
}

public interface Processor<T, R> extends Subscriber<T>, Publisher<R> {}
```

How it works?



Reactive Streams Implementations for Java:

RxJava

Project Reactor

Akka Streams

Ratpack

Vert.x 3

Spring Framework 5

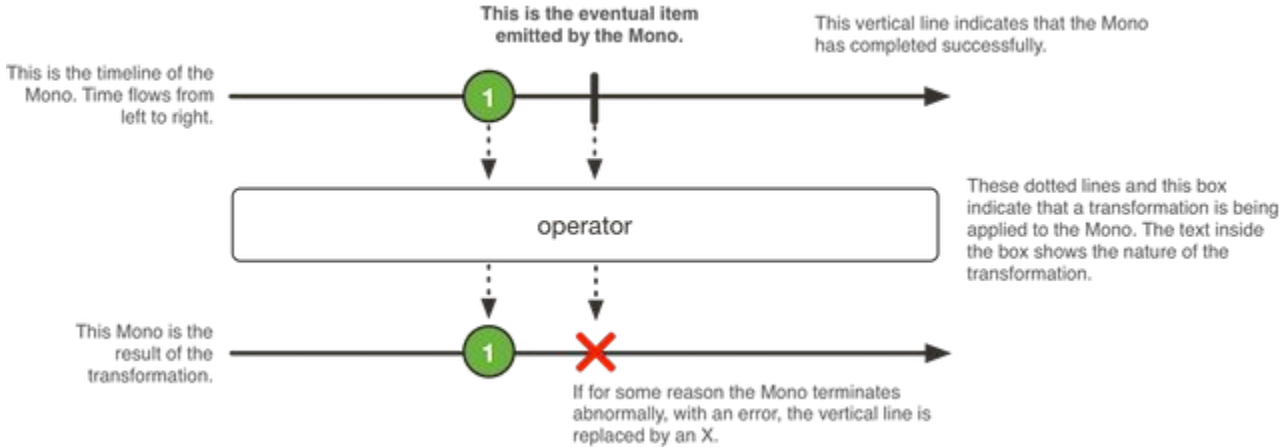
Another major release, became GA in September 2017

A lots of improvements and new concepts introduced:

- Support for JDK 9
- Support Java EE 8 API (e.g. Servlet 4.0)
- Integration with Project Reactor 3.1
- JUnit 5
- Comprehensive support for Kotlin language
- Dedicated reactive web framework - Spring WebFlux

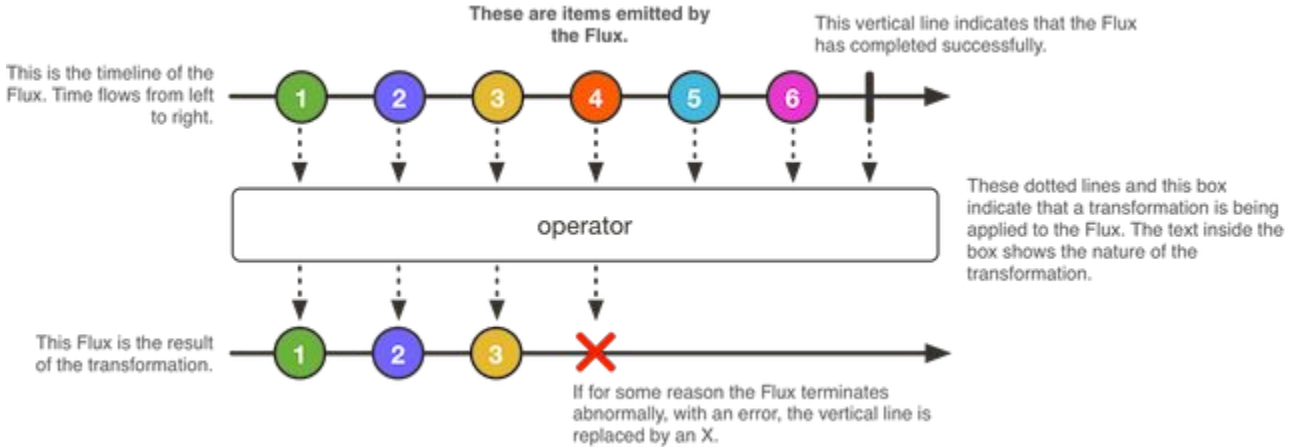
Project Reactor: `Mono<T>`

`Publisher` which emits **0** or **1** element (successfully or with an error)

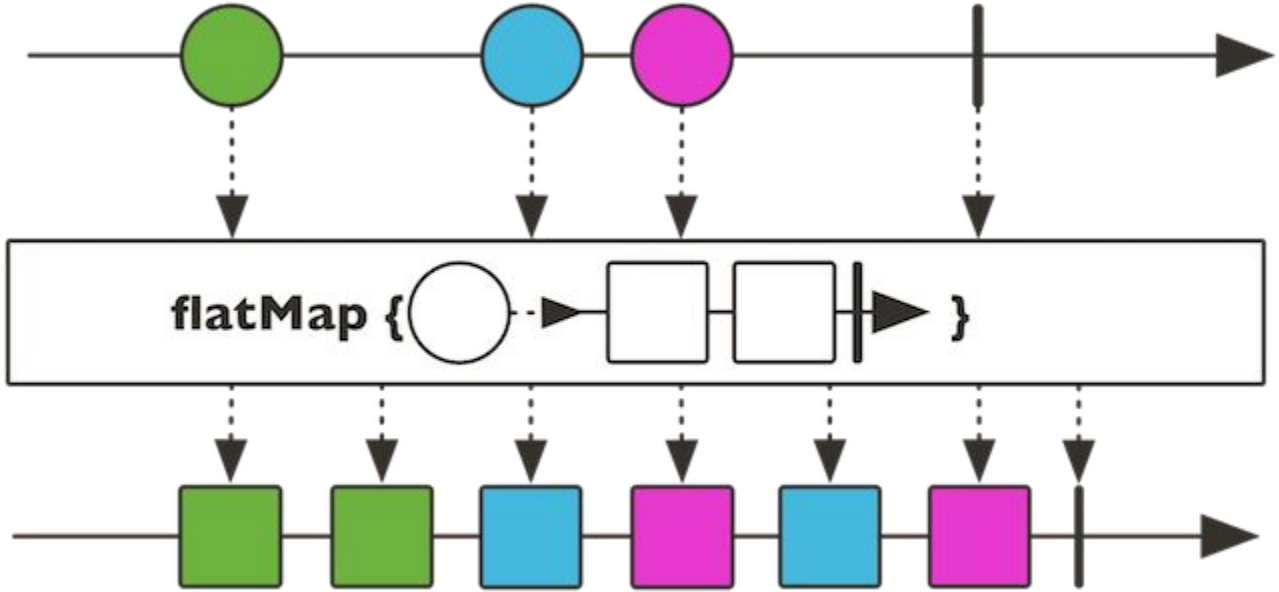


Project Reactor: `Flux<T>`

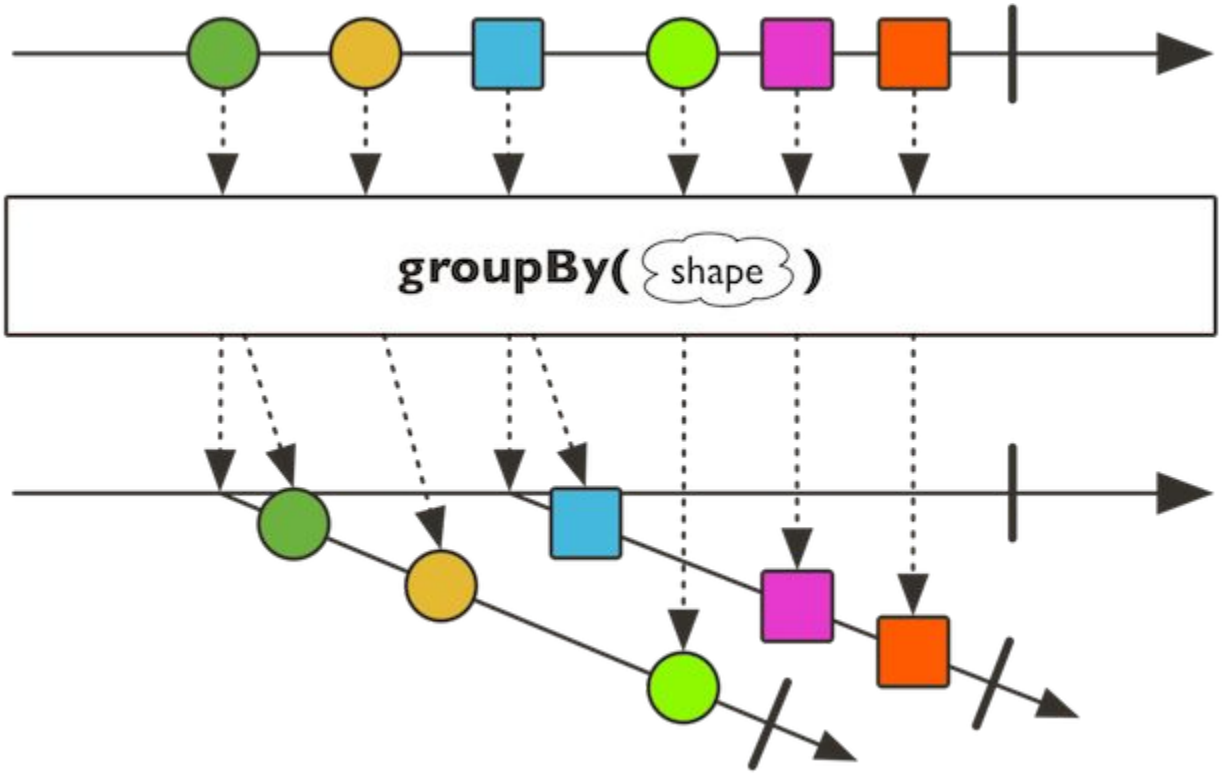
`Publisher` which emits **0** to **N** elements (successfully or with an error)



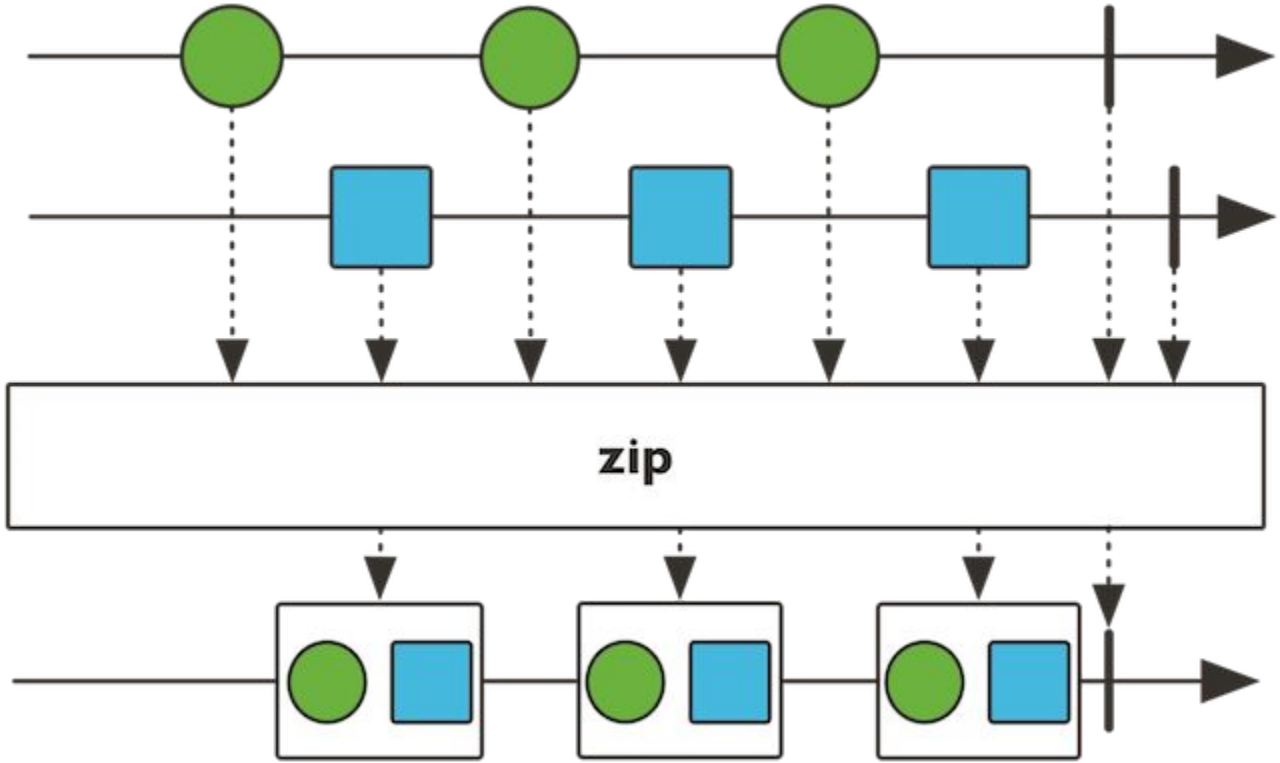
Various Reactor Operators



Various Reactor Operators



Various Reactor Operators

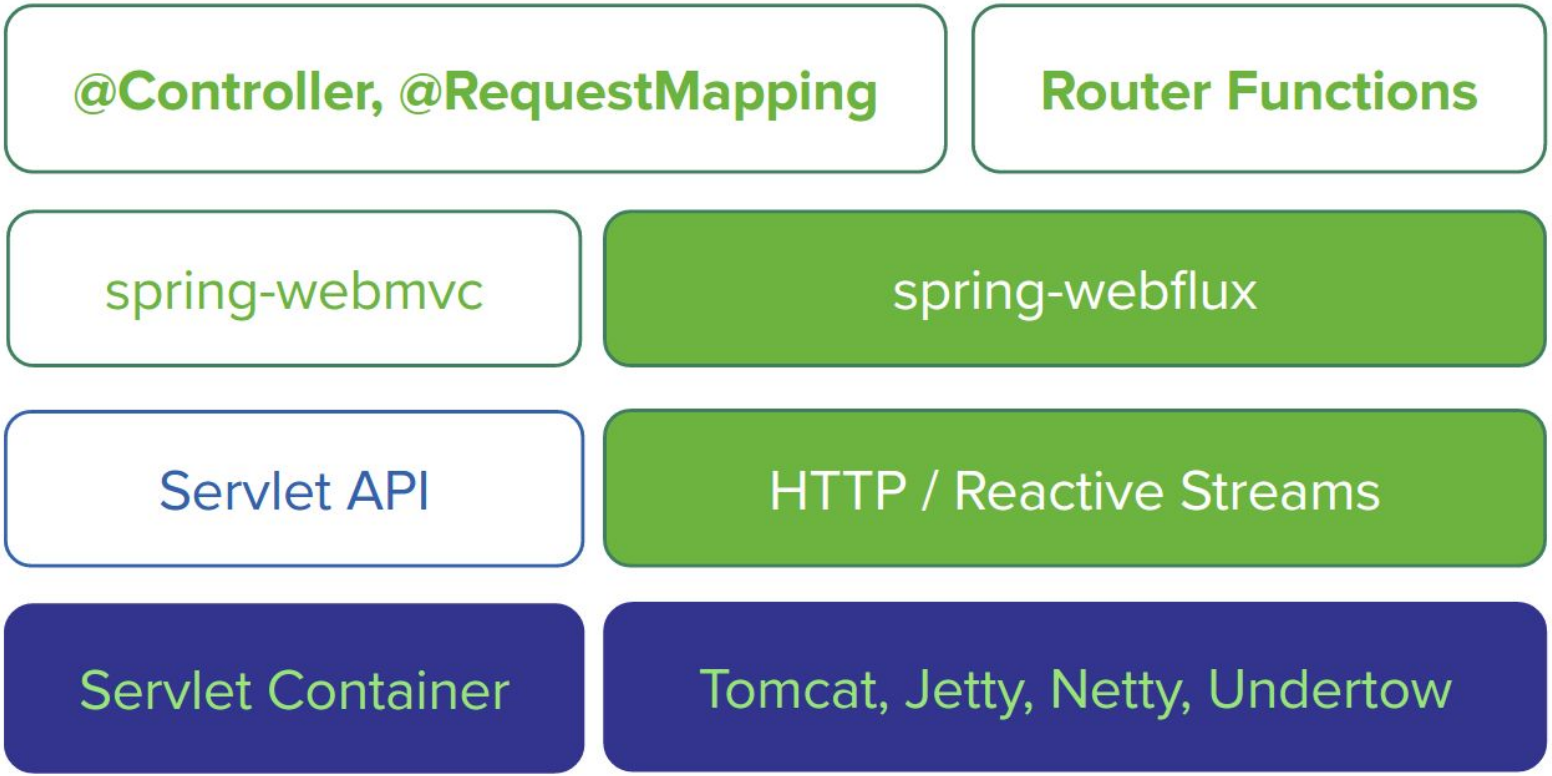


Reactor Pipeline

```
userService.getFavorites(userId)
    .timeout(Duration.ofMillis(800))
    .onErrorResume(cacheService.cachedFavoritesFor(userId))
    .flatMap(favoriteService::getDetails)
    .switchIfEmpty(suggestionService.getSuggestions())
    .take(5)
    .publishOn(UiUtils.uiThreadScheduler())
    .subscribe(uiList::show, UiUtils::errorPopup);
```

- Lazy evaluated
- Nothing is produced until there is a subscriber

Spring 5 Reactive Web



Annotation-based Programming Model

```
@RestController
public class PersonController {
    private final PersonRepository repository;
    public PersonController(PersonRepository repository) {
        this.repository = repository;
    }

    @GetMapping("/person")
    Flux<Person> list() {
        return this.repository.findAll();
    }

    @GetMapping("/person/{id}")
    Mono<Person> findById(@PathVariable String id) {
        return this.repository.findOne(id);
    }
}
```

Functional Programming Model - Handler

```
public class PersonHandler {
  ...
  public Mono<ServerResponse> listPeople(ServerRequest request) {
    Flux<Person> people = repository.allPeople();
    return ServerResponse.ok().contentType(APPLICATION_JSON).body(people, Person.class);
  }

  public Mono<ServerResponse> createPerson(ServerRequest request) {
    Mono<Person> person = request.bodyToMono(Person.class);
    return ServerResponse.ok().build(repository.savePerson(person));
  }

  public Mono<ServerResponse> getPerson(ServerRequest request) {
    int personId = Integer.valueOf(request.pathVariable("id"));
    Mono<ServerResponse> notFound = ServerResponse.notFound().build();
    Mono<Person> personMono = this.repository.getPerson(personId);
    return personMono
          .then(person ->
                ServerResponse.ok().contentType(APPLICATION_JSON).body(fromObject(person)))
          .otherwiseIfEmpty(notFound);
  }
}
```

Functional Programming Model - Router

```
PersonRepository repository = ...
PersonHandler handler = new PersonHandler(repository);

RouterFunction<ServerResponse> personRoute =
  route(GET("/person/{id}").and(accept(APPLICATION_JSON)), handler::getPerson)
    .andRoute(GET("/person").and(accept(APPLICATION_JSON)), handler::listPeople)
    .andRoute(POST("/person").and(contentType(APPLICATION_JSON)), handler::createPerson);

// Run in Reactor Netty
HttpHandler httpHandler = RouterFunctions.toHttpHandler(route);
ReactorHttpHandlerAdapter adapter = new ReactorHttpHandlerAdapter(httpHandler);
HttpServer server = HttpServer.create(HOST, PORT);
server.newHandler(adapter).block();

// Run in Tomcat
HttpHandler httpHandler = RouterFunctions.toHttpHandler(route);
HttpServlet servlet = new ServletHttpHandlerAdapter(httpHandler);
Tomcat server = new Tomcat();
Context rootContext = server.addContext("", System.getProperty("java.io.tmpdir"));
Tomcat.addServlet(rootContext, "servlet", servlet);
rootContext.addServletMapping("/", "servlet");
tomcatServer.start();
```

Functional Reactive Client

```
WebClient client = WebClient.create("http://example.com");

Mono<Account> account = client.get()
   .url("/accounts/{id}", 1L)
   .accept(APPLICATION_JSON)
   .exchange(request)
   .then(response -> response.bodyToMono(Account.class));
```

Functional Reactive WebSocket Client

```
WebSocketClient webSocketClient = new ReactorNettyWebSocketClient();
webSocketClient.execute(new URI("wss://echo.websocket.org"),
  session -> session.send(input.map(session::textMessage))
    .thenMany(session
      .receive()
      .map(WebSocketMessage::getPayloadAsText)
      .log())
    .then())
```

Spring Data Reactive

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-mongodb-reactive</artifactId>
</dependency>
```

```
public interface BookRepository
    extends ReactiveCrudRepository<Book, String> {

  Flux<Book> findByAuthor(String author);
}
```

WebFlux Spring Security

```
@EnableWebFluxSecurity
public class HelloWebfluxSecurityConfig {

    @Bean
    public MapReactiveUserDetailsService userDetailsService() {
        UserDetails user = User.withDefaultPasswordEncoder()
            .username("user")
            .password("user")
            .roles("USER")
            .build();
        return new MapReactiveUserDetailsService(user);
    }
}
```

Reactive Method Security

```
@EnableWebFluxSecurity
@EnableReactiveMethodSecurity
public class SecurityConfig {
    @Bean
    public MapReactiveUserDetailsService userDetailsService() {...}
}

@Component
public class HelloWorldMessageService {
    @PreAuthorize("hasRole('ADMIN')")
    public Mono<String> findMessage() {
        return Mono.just("Hello World!");
    }
}
```

DEMO

https://github.com/drazen-nikolic/reactive-spring-5-demo

Questions?

References & Attributions

Reactive Streams Specification for the JVM

Reactive Spring - Josh Long, Mark Heckler

Reactive Programming by Venkat Subramaniam

What is Reactive Programming by Martin Oderski

Reactive Streams: Handling Data-Flow the Reactive Way by Roland Kuhn

What Are Reactive Streams in Java? by John Thompson

Spring Boot Reactive Tutorial by Mohit Sinha

Doing Reactive Programming with Spring 5 by Eugen Paraschiv

Where applicable...

Be proactive, go Reactive!

***Spring** will help you on this journey!*

Thank you