Part II -

Unknown macro: 'seo-metadata'

Introduction

spring.io Notes on Reactive Programming Part II: Writing Some Code . Java Reactor , . Reactor Java .

-
- (Setting up a Project)
- ? (What makes it functional?)
- Conclusions

Prerequisite

.

- ( )
- JSDK >= 8
- Maven / Gradle
- Spring Boot
- Java Generic
- Functional Interface Lambda Expression
-

, . , , . , API , . , (flow of data) .

(Setting up a Project)

, Reactor . . Copy and Paste Github .

https://start.spring.io/ Reactor Core . Maven .

reactor-core Maven dependency

```
<dependency>
        <groupId>io.projectreactor</groupId>
        <artifactId>reactor-core</artifactId>
        <version>3.0.7.RELEASE</version><!--$NO-MVN-MAN-VER$-->
</dependency>
```

Gradle .

reactor-core Gradle dependency

```
compile 'io.projectreactor:reactor-core:3.0.7.RELEASE'
```

.

? (What makes it functional?)

(sequence of events) (publisher) (subscriber) . (sequence) stream - - . stream s stream , Java 8 java.util.Stream . . (Reactive Stream .)

Reactor , Reactor . Reactor *Flux* .(Reactive Stream Publisher .) RxJava , *Flux* *Observable* .(Reactor 2.0 Stream Java 8 Stream . Reactor 3.0 .)

(, Generators)

Flux POJO , generic . , Flux<T> T . *Flux* (*Flux*) static . *Flux* .

Creating Flux from String array

```
Flux<String> flux = Flux.just("red", "white", "blue");
```

Flux , . : ( ), (.).

(Single Valued Sequences)

, (repository) id (enitty) , (element) . Reactor *Mono* .*Mono* *Flux* API *Flux* , *Flux* (operator) . RxJava (version 1.x) *Single* , *Completable* . Reactor *Mono<Void>* .

(Operators)

Flux , (operator). , (javadoc) . .

, *Flux* (standard out) .log() . , (transform) map() :

using Flux operators

```
Flux<String> flux = Flux.just("red", "white", "blue");

Flux<String> upper = flux
  .log()
  .map(String::toUpperCase);
```

(convert) (transform). , .

- - . , . (.) *Flux* (plan of execution) . (declarative), "" . (flow) , (*Publisher*) *Flux* (subscribe) .

, , Java 8 *Streams* . Flux *Stream* .

Java8 Stream

```
Stream<String> stream = Streams.of("red", "white", "blue");
Stream<String> upper = stream.map(value -> {
    System.out.println(value);
    return value.toUpperCase();
});
```

Flux : , . *Flux* *Stream* , *Stream* . *Flux* , , - - (comsume) .

Sebastien Deleuze Reactive Types , streaming API ( ) . *Flux* *Stream* .

(Subscribers)

(flow) , subscribe() Flux . . subscribe (chain) . , (iterated). , , HTTP .

subscribe() .

subscribing Flux Sequence

```
Flux.just("red", "white", "blue")
  .log()
  .map(String::toUpperCase)
.subscribe();
```

.

logs on standard out

```
09:17:59.665 [main] INFO reactor.core.publisher.FluxLog -  onSubscribe(reactor.core.publisher.
FluxIterable$IterableSubscription@3ffc5af1)
09:17:59.666 [main] INFO reactor.core.publisher.FluxLog -  request(unbounded)
09:17:59.666 [main] INFO reactor.core.publisher.FluxLog -  onNext(red)
09:17:59.667 [main] INFO reactor.core.publisher.FluxLog -  onNext(white)
09:17:59.667 [main] INFO reactor.core.publisher.FluxLog -  onNext(blue)
09:17:59.667 [main] INFO reactor.core.publisher.FluxLog -  onComplete()
```

, *subscribe()* . *request()* , "unbounded" . , (*onNext()*), (*onComplete()*), (*onSubscribe()*) . *doOn*()* , .

subscribe() , . *subscribe()* . *Consumer*, , *Consumer* , *Runnable* . :

callback for per-item

```
 Flux.just("red", "white", "blue")
    .log()
    .map(String::toUpperCase)
.subscribe(System.out::println);
```

.

Output for per-item callback

```
09:56:12.680 [main] INFO reactor.core.publisher.FluxLog -  onSubscribe(reactor.core.publisher.
FluxArray$ArraySubscription@59f99ea)
09:56:12.682 [main] INFO reactor.core.publisher.FluxLog -  request(unbounded)
09:56:12.682 [main] INFO reactor.core.publisher.FluxLog -  onNext(red)
RED
09:56:12.682 [main] INFO reactor.core.publisher.FluxLog -  onNext(white)
WHITE
09:56:12.682 [main] INFO reactor.core.publisher.FluxLog -  onNext(blue)
BLUE
09:56:12.682 [main] INFO reactor.core.publisher.FluxLog -  onComplete()
```

, , () "(bound)" . *Subscription* API *Subscriber* . , *subscribe()* .

long form of subscription

```
.subscribe(new Subscriber<String>() {

    @Override
    public void onSubscribe(Subscription s) {
        s.request(Long.MAX_VALUE);
    }
    @Override
    public void onNext(String t) {
        System.out.println(t);
    }
    @Override
    public void onError(Throwable t) {
    }
    @Override
    public void onComplete() {
    }

});
```

, , *Subscription* .

execute at most 2 items at a time

```
.subscribe(new Subscriber<String>() {

    private long count = 0;
    private Subscription subscription;

    @Override
    public void onSubscribe(Subscription subscription) {
        this.subscription = subscription;
        subscription.request(2);
    }

    @Override
    public void onNext(String t) {
        count++;
        if (count>=2) {
            count = 0;
            subscription.request(2);
        }
     }
...
```

Subscriber . , . .

output for batching 2 items

```
09:47:13.562 [main] INFO reactor.core.publisher.FluxLog -  onSubscribe(reactor.core.publisher.
FluxArray$ArraySubscription@61832929)
09:47:13.564 [main] INFO reactor.core.publisher.FluxLog -  request(2)
09:47:13.564 [main] INFO reactor.core.publisher.FluxLog -  onNext(red)
09:47:13.565 [main] INFO reactor.core.publisher.FluxLog -  onNext(white)
09:47:13.565 [main] INFO reactor.core.publisher.FluxLog -  request(2)
09:47:13.565 [main] INFO reactor.core.publisher.FluxLog -  onNext(blue)
09:47:13.565 [main] INFO reactor.core.publisher.FluxLog -  onComplete()
```

, *Flux* . .

batching with Flux method

```
Flux.just("red", "white", "blue")
  .log()
  .map(String::toUpperCase)
.subscribe(null, 2);
```

(subscribe() (request limit) .) .

batching subscription result

```
10:25:43.739 [main] INFO reactor.core.publisher.FluxLog -  onSubscribe(reactor.core.publisher.
FluxArray$ArraySubscription@4667ae56)
10:25:43.740 [main] INFO reactor.core.publisher.FluxLog -  request(2)
10:25:43.740 [main] INFO reactor.core.publisher.FluxLog -  onNext(red)
10:25:43.741 [main] INFO reactor.core.publisher.FluxLog -  onNext(white)
10:25:43.741 [main] INFO reactor.core.publisher.FluxLog -  request(2)
10:25:43.741 [main] INFO reactor.core.publisher.FluxLog -  onNext(blue)
10:25:43.741 [main] INFO reactor.core.publisher.FluxLog -  onComplete()
```

Reactive , (subscription) . , , . , , (processing layer) .

, , (Threads, Schedulers, and Background Processing)

"main" , *subscribe()* main . : Reactor , . 5, , . : JVM , , . Reactor , .

Flux . , Flux.subscribeOn() :

subscription on background thread

```
Flux.just("red", "white", "blue")
  .log()
  .map(String::toUpperCase)
  .subscribeOn(Schedulers.parallel())
.subscribe(null, 2);
```

.

output for parallel subscribing

```
13:43:41.279 [parallel-1-1] INFO reactor.core.publisher.FluxLog -  onSubscribe(reactor.core.publisher.
FluxArray$ArraySubscription@58663fc3)
13:43:41.280 [parallel-1-1] INFO reactor.core.publisher.FluxLog -  request(2)
13:43:41.281 [parallel-1-1] INFO reactor.core.publisher.FluxLog -  onNext(red)
13:43:41.281 [parallel-1-1] INFO reactor.core.publisher.FluxLog -  onNext(white)
13:43:41.281 [parallel-1-1] INFO reactor.core.publisher.FluxLog -  request(2)
13:43:41.281 [parallel-1-1] INFO reactor.core.publisher.FluxLog -  onNext(blue)
13:43:41.281 [parallel-1-1] INFO reactor.core.publisher.FluxLog -  onComplete()
```

Copy and Paste , JVM (wait) .

(parallel-1-1) - Flux . CPU .(, .) , I/O , I/O . , . , *Schedulers.parallel()* . ( ) , *Flux* , . *flatMap()* , ( ) *Publisher* , :

flatMap with parallel execution example

```
Flux.just("red", "white", "blue")
  .log()
  .flatMap(value ->
     Mono.just(value.toUpperCase())
       .subscribeOn(Schedulers.parallel()),
     2)
.subscribe(value -> {
  log.info("Consumed: " + value);
})
```

, *flatMap()* "" , . Reactor , . , . (Reactive Gems .)

.

parallel processing result

```
15:24:36.596 [main] INFO reactor.core.publisher.FluxLog -  onSubscribe(reactor.core.publisher.
FluxIterable$IterableSubscription@6f1fba17)
15:24:36.610 [main] INFO reactor.core.publisher.FluxLog -  request(2)
15:24:36.610 [main] INFO reactor.core.publisher.FluxLog -  onNext(red)
15:24:36.613 [main] INFO reactor.core.publisher.FluxLog -  onNext(white)
15:24:36.613 [parallel-1-1] INFO com.example.FluxFeaturesTests - Consumed: RED
15:24:36.613 [parallel-1-1] INFO reactor.core.publisher.FluxLog -  request(1)
15:24:36.613 [parallel-1-1] INFO reactor.core.publisher.FluxLog -  onNext(blue)
15:24:36.613 [parallel-1-1] INFO reactor.core.publisher.FluxLog -  onComplete()
15:24:36.614 [parallel-3-1] INFO com.example.FluxFeaturesTests - Consumed: BLUE
15:24:36.617 [parallel-2-1] INFO com.example.FluxFeaturesTests - Consumed: WHITE
```

, , *flatMap()* . *request(1)* , . Reactor , Reactor *Publisher* (pre-fetch).(.)

("red", "white", "blue") . .

Flux , *publishOn()* , . (, *onNext()* .) :

callback for listeners

```
Flux.just("red", "white", "blue")
  .log()
  .map(String::toUpperCase)
  .subscribeOn(Schedulers.newParallel("sub"))
  .publishOn(Schedulers.newParallel("pub"), 2)
.subscribe(value -> {
    log.info("Consumed: " + value);
});
```

.

result of publishOn

```
15:12:09.750 [sub-1-1] INFO reactor.core.publisher.FluxLog -  onSubscribe(reactor.core.publisher.
FluxIterable$IterableSubscription@172ed57)
15:12:09.758 [sub-1-1] INFO reactor.core.publisher.FluxLog -  request(2)
15:12:09.759 [sub-1-1] INFO reactor.core.publisher.FluxLog -  onNext(red)
15:12:09.759 [sub-1-1] INFO reactor.core.publisher.FluxLog -  onNext(white)
15:12:09.770 [pub-1-1] INFO com.example.FluxFeaturesTests - Consumed: RED
15:12:09.771 [pub-1-1] INFO com.example.FluxFeaturesTests - Consumed: WHITE
15:12:09.777 [sub-1-1] INFO reactor.core.publisher.FluxLog -  request(2)
15:12:09.777 [sub-1-1] INFO reactor.core.publisher.FluxLog -  onNext(blue)
15:12:09.777 [sub-1-1] INFO reactor.core.publisher.FluxLog -  onComplete()
15:12:09.783 [pub-1-1] INFO com.example.FluxFeaturesTests - Consumed: BLUE
```

("Consumed:...") pub-1-1 . *subscribeOn()* (chunk) pub-1-1 . , Reactor , .

subscribe(null, 2) publishOn() prefetch=2 . subscribe() fetchSize .

subscribeOn() .

without subscribeOn

```
15:38:52.191 [main] DEBUG reactor.util.Loggers$LoggerFactory - Using Slf4j logging framework
15:38:52.229 [main] INFO reactor.Flux.Array.1 - | onSubscribe([Synchronous Fuseable] FluxArray.
ArraySubscription)
15:38:52.243 [main] INFO reactor.Flux.Array.1 - | request(2)
15:38:52.244 [main] INFO reactor.Flux.Array.1 - | onNext(red)
15:38:52.244 [main] INFO reactor.Flux.Array.1 - | onNext(white)
15:38:52.244 [pub-1] DEBUG com.sample.demo.Main - Consumed: RED
15:38:52.244 [pub-1] DEBUG com.sample.demo.Main - Consumed: WHITE
15:38:52.244 [pub-1] INFO reactor.Flux.Array.1 - | request(2)
15:38:52.244 [pub-1] INFO reactor.Flux.Array.1 - | onNext(blue)
15:38:52.245 [pub-1] INFO reactor.Flux.Array.1 - | onComplete()
15:38:52.245 [pub-1] DEBUG com.sample.demo.Main - Consumed: BLUE
```

: (Extractors: The Subscribers from Dark Side)

, *Mono.block() Mono.toFuture()*, *Flux.toStream()* .("" , .) *Flux collectList() collectMap()* (converter) , Flux Mono . , .

" " . (.), . .

; Spring MVC API . *Mono.block()* Reactive Stream . Reactive Stream Java 8 *Streams* . Java *Stream* " " , *Mono.block()* . *subscribe()* , , .- subscribeOn() , .

Conclusions

Reactive Stream Reactor API . , . ( ) GitHub , Lite RX Hands On . , , . , , , .

- Part I - (SYS4U OPEN WIKI)
  - spring
  - reactive
  - reactor
- Part II - (SYS4U OPEN WIKI)

References

- https://spring.io/blog/2016/06/13/notes-on-reactive-programming-part-ii-writing-some-code
- https://spring.io/blog/2016/04/19/understanding-reactive-types

Related Information

- https://start.spring.io/

  - reactive
  - reactor
  - spring
- (Reactive Programming) (SYS4U OPEN WIKI)
  - reactive
  - menifesto
- Reactor (SYS4U OPEN WIKI)
  - reactor
  - tutorial
  - techio
  - stepverifier
- Reactive Programming with Spring Boot (SYS4U OPEN WIKI)
  - springboot
  - reactive
  - mongo-db
  - gradle
  - thymeleaf
  - jsdk
  - lombok
  - lambda
  - functionalinterface
- 1. Reactive Programming (SYS4U OPEN WIKI)
  - reactive
  - springboot
  - rp
  - asynchronous
  - non-blocking
  - immutable
  - fp
  - functional-programming
  - frp
  - functional-reactive-programming
- Kotlin with Springboot (SYS4U OPEN WIKI)
  - kotlin
  - spring
  - springboot
- MyBatis @Mapper ? (SYS4U OPEN WIKI)
  - beandefinition
  - beandefinitionregistry
  - proxy
  - spring
  - mybatis