

# Webflux

## 5 Reactor 이론

# Reactive Stream

(specification)

# Reactive Stream

(specification)

Spring Webflux

Reactor

Reactive  
Stream



## Reactive Streams

Reactive Streams is an initiative to provide a standard for asynchronous stream processing with non-blocking back pressure. This encompasses efforts aimed at runtime environments (JVM and JavaScript) as well as network protocols.

### JDK9 `java.util.concurrent.Flow`

The interfaces available in JDK >= 9 [java.util.concurrent.Flow](#), are 1:1 semantically equivalent to their respective Reactive Streams counterparts. This means that there will be a migratory period, while libraries move to adopt the new types in the JDK, however this period is expected to be short - due to the full semantic equivalence of the libraries, as well as the Reactive Streams <-> Flow adapter library as well as a TCK compatible directly with the JDK Flow types.

Read [this](#) if you are interested in learning more about Reactive Streams for the JVM.

### The Problem

Handling streams of data—especially “live” data whose volume is not predetermined—requires special care in an asynchronous system. The most

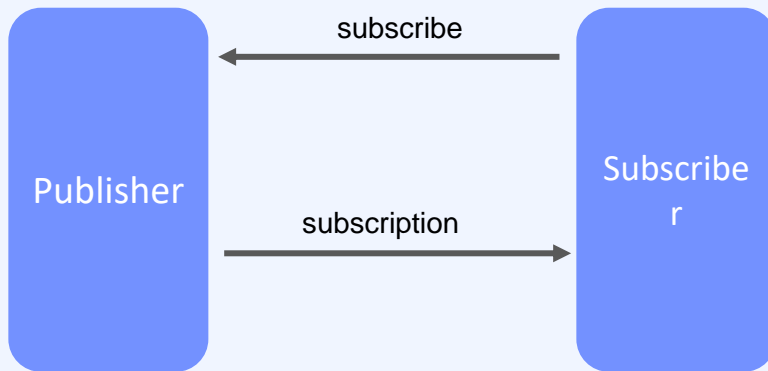
<https://www.reactive-streams.org/>

## 1. stream

- a. publisher
- b. subscriber
- c. subscription
- d. processor

2. asynchronous

3. back pressure

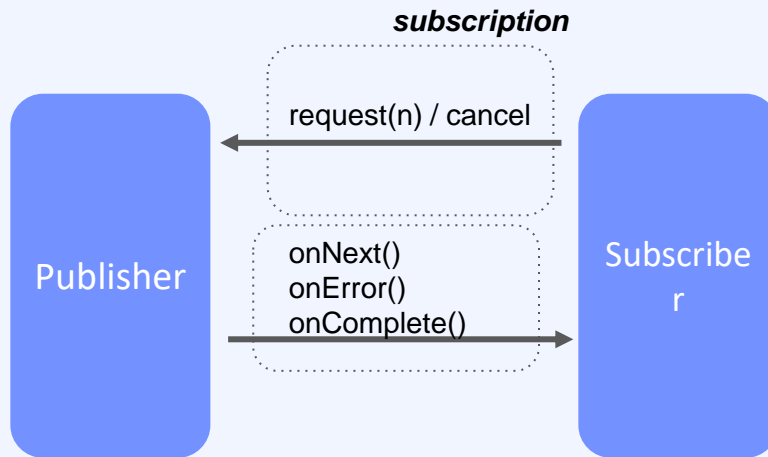


## 1. stream

- a. publisher
- b. subscriber
- c. subscription
- d. processor

2. asynchronous

3. back pressure

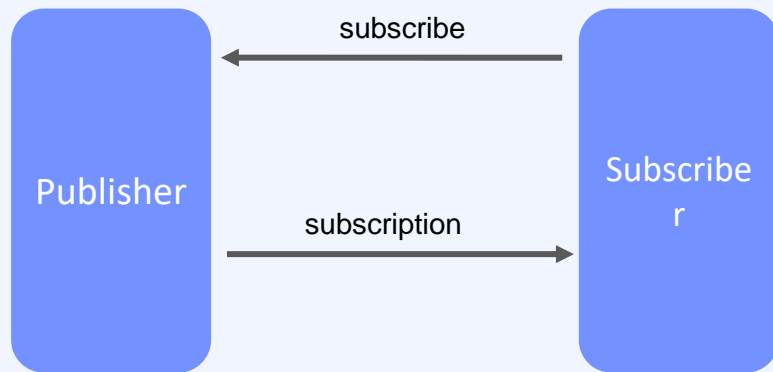


## Reactor

### 구성요소

3.

Webflux



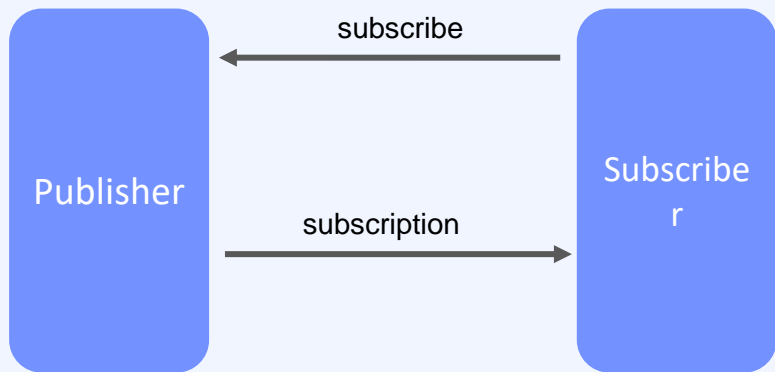
```
public interface Publisher<T> {  
    public void subscribe(Subscriber<? super T> s);  
}
```

## Reactor

### 구성요소

3.

Webflux



```
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();  
}
```

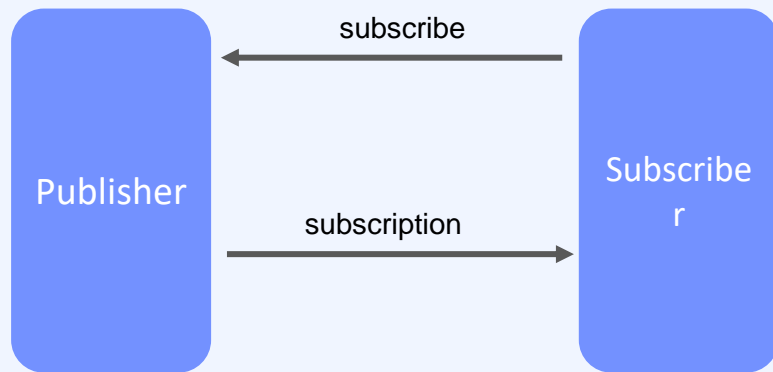


## Reactor

### 구성요소

3.

Webflux



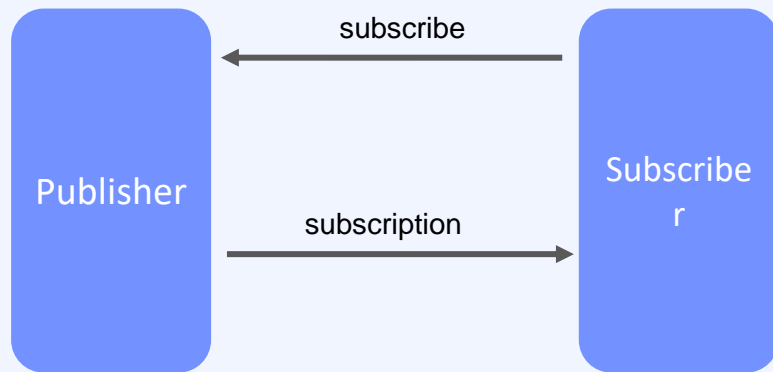
```
public interface Subscription {  
    public void request(long n);  
    public void cancel();  
}
```

## Reactor

### 구성요소

3.

Webflux



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

## Spring Webflux

### 주요 특징

3.

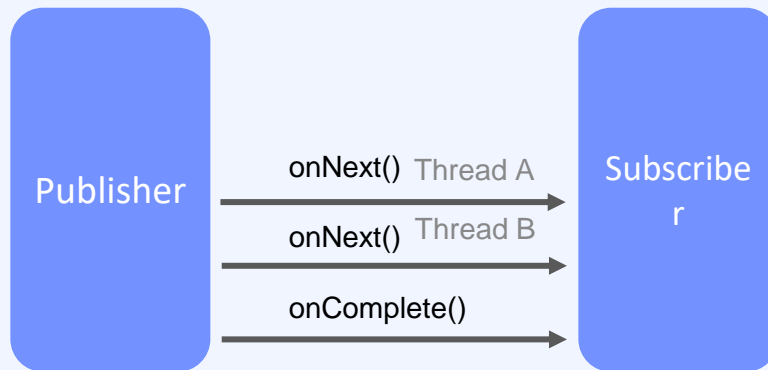
Webflux

#### 1. stream

- a. publisher
- b. subscriber
- c. subscription
- d. processor

#### 2. asynchronous

#### 3. back pressure



## Spring Webflux

### 주요 특징

3.

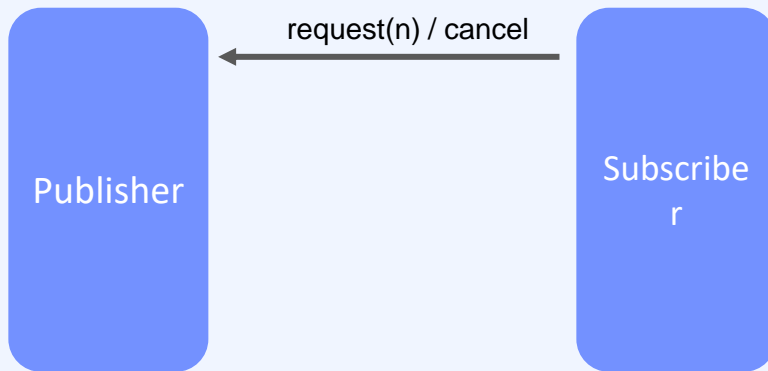
Webflux

#### 1. stream

- a. publisher
- b. subscriber
- c. subscription
- d. processor

#### 2. asynchronous

#### 3. back pressure



## Spring Webflux

### 주요 특징

3.

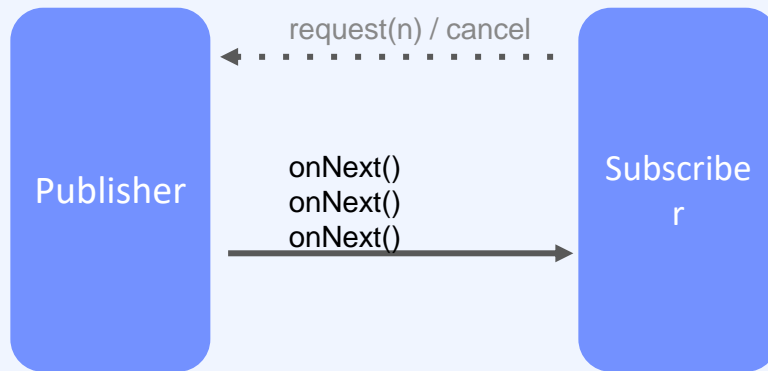
Webflux

#### 1. stream

- a. publisher
- b. subscriber
- c. subscription
- d. processor

#### 2. asynchronous

#### 3. back pressure



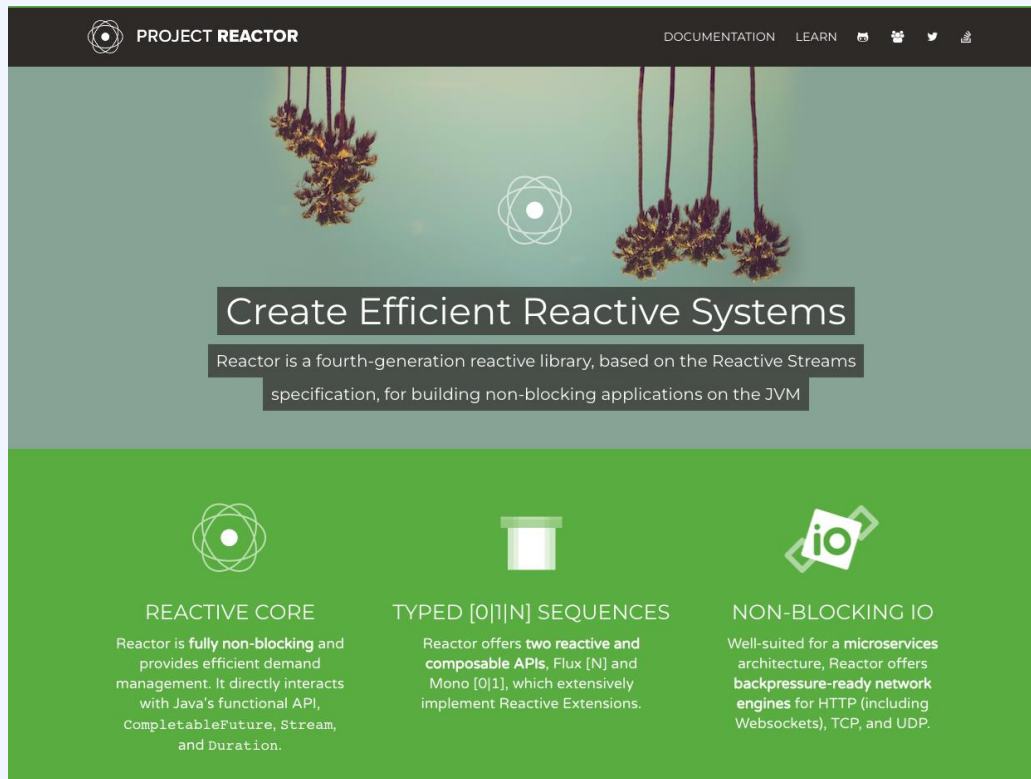
# Project reactor

## Spring Webflux

reactor

3.

Webflux



The image is a screenshot of the Project Reactor website. The header is dark with the Project Reactor logo and navigation links for 'DOCUMENTATION' and 'LEARN'. The main content area has a green background with a central text box that reads 'Create Efficient Reactive Systems'. Below this, a paragraph states: 'Reactor is a fourth-generation reactive library, based on the Reactive Streams specification, for building non-blocking applications on the JVM'. The bottom section is divided into three columns, each with an icon and a title: 'REACTIVE CORE' (with the Reactor logo icon), 'TYPED [0|1|N] SEQUENCES' (with a box icon), and 'NON-BLOCKING IO' (with an 'io' icon). Each column contains a brief description of its respective feature.

PROJECT REACTOR

DOCUMENTATION LEARN

# Create Efficient Reactive Systems

Reactor is a fourth-generation reactive library, based on the Reactive Streams specification, for building non-blocking applications on the JVM

### REACTIVE CORE

Reactor is **fully non-blocking** and provides efficient demand management. It directly interacts with Java's functional API, `CompletableFuture`, `Stream`, and `Duration`.

### TYPED [0|1|N] SEQUENCES

Reactor offers **two reactive and composable APIs**, `Flux [N]` and `Mono [0|1]`, which extensively implement Reactive Extensions.

### NON-BLOCKING IO

Well-suited for a **microservices** architecture, Reactor offers **backpressure-ready network engines** for HTTP (including Websockets), TCP, and UDP.

<https://projectreactor.io/>

## 마무리

3.

Webflux

1. Reactive Stream
  - a. Asynchronous Stream processing
  - b. Nonblocking backpressure
2. Publisher, Subscriber, Subscription, Processor
3. Reactor