Mono class in Java: what is, and when to use?

Asked 4 years, 9 months ago Modified 1 year, 10 months ago Viewed 96k times

65

I have this following code:

```
import org.springframework.http.MediaType;
import org.springframework.stereotype.Component;
import org.springframework.web.reactive.function.BodyInserters;
import org.springframework.web.reactive.function.server.ServerRequest;
import org.springframework.web.reactive.function.server.ServerResponse;
import reactor.core.publisher.Mono;

@Component
public class GreetingHandler
    public Mono<ServerResponse> hello(ServerRequest request) {
        return ServerResponse.ok().contentType(MediaType.TEXT_PLAIN)
        .body(BodyInserters.fromValue("Hello Spring!"));
    }
}
```

I understand this code except what the class Mono does and what are its features. I did a lot of search but it didn't goes straight to the point: **what is the class Mono** and **when to use it**?

java spring spring-boot project-reactor spring-mono

Share Edit Follow

edited Jul 27, 2020 at 10:30

Daniel Fath
17.9k ● 8 ● 49 ● 83

asked Mar 16, 2020 at 11:45

S.I.M.P.L.E
795 ● 1 ● 5 ● 10

projectreactor.io/docs/core/release/api/reactor/core/publisher/... – Michael Mar 16, 2020 at 11:47

1 Answer

Sorted by: Highest score (default)

75

A `Mono<T>` is a specialized `Publisher<T>` that emits at most one item and then (optionally) terminates with an `onComplete` signal or an `onError` signal. It offers only a subset of the operators that are available for a `Flux`, and some operators (notably those that combine the `Mono` with another `Publisher`) switch to a `Flux`. For example, `Mono#concatWith(Publisher)` returns a `Flux` while `Mono#then(Mono)` returns another `Mono`. Note that you can use a `Mono` to represent no-value asynchronous processes that only have the concept of completion (similar to a Runnable). To create one, you can use an empty `Mono<Void>`.

Mono and Flux are both reactive streams. They differ in what they express. A Mono is a stream of 0 to 1 element, whereas a Flux is a stream of 0 to N elements.

This difference in the semantics of these two streams is very useful, as for example making a request to an Http server expects to receive 0 or 1 response, it would be inappropriate to use a Flux in this case. On the opposite, computing the result of a mathematical function on an interval expects one result per number in the interval. In this other case, using a Flux is appropriate.

How to use it:

```
Mono.just("Hello World !").subscribe(
  successValue -> System.out.println(successValue),
  error -> System.err.println(error.getMessage()),
  () -> System.out.println("Mono consumed.")
);
// This will display in the console :
// Hello World !
// Mono consumed.

// In case of error, it would have displayed :
// **the error message**
// Mono consumed.

Flux.range(1, 5).subscribe(
  successValue -> System.out.println(successValue),
  error -> System.err.println(error.getMessage()),
  () -> System.out.println("Flux consumed.")
);
// This will display in the console :
// 1
// 2
// 3
// 4
// 5
// Flux consumed.

// Now imagine that when manipulating the values in the Flux, an exception
// is thrown for the value 4.
// The result in the console would be :
// An error as occurred
// 1
// 2
// 3
//
// As you can notice, the "Flux consumed." doesn't display because the Flux
// hasn't been fully consumed. This is because the stream stop handling future
values
// if an error occurs. Also, the error is handled before the successful values.
```

sources: Reactor Java #1 - How to create Mono and Flux?, Mono, an Asynchronous 0-1 Result

it might be helpful: Mono doc

Share Edit Follow edited Feb 2, 2023 at 23:46 answered Mar 16, 2020 at 13:18

Thomas Turrell-Croft
712 ● 7 ● 8

Alessandro Candeloro
1,061 ● 6 ● 20