Comparing & Contrasting
Spring WebMVC & WebFlux

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Lesson

- Recognize the similarities & differences between Spring WebMVC & WebFlux frameworks supported by Spring Boot 2.0

---

**Reactive Stack**
Spring WebFlux is a non-blocking web framework built from the ground up to take advantage of multi-core, next-generation processors and handle massive numbers of concurrent connections.

- Netty, Servlet 3.1+ Containers
- Reactive Streams Adapters
- Spring Security Reactive
- Spring WebFlux

**Spring Data Reactive Repositories**
Mongo, Cassandra, Redis, Couchbase

**Servlet Stack**
Spring MVC is built on the Servlet API and uses a synchronous blocking I/O architecture with a one-request-per-thread model.

- Servlet Containers
- Servlet API
- Spring Security
- Spring MVC

**Spring Data Repositories**
JDBC, JPA, NoSQL
Comparing & Contrasting Spring WebMVC & WebFlux
Comparing & Contrasting Spring WebMVC & WebFlux

- Spring WebMVC & WebFlux have similarities & differences wrt functionality & internal components

See maddy4java.blogspot.com/2019/11/spring-boot-spring-webflux-vs-spring-mvc.html
Comparing & Contrasting Spring WebMVC & WebFlux

- WebMVC is sync

Built on Servlet API & uses a sync I/O w/one-thread-per-request model (by default)
Comparing & Contrasting Spring WebMVC & WebFlux

• WebMVC is sync
• The server uses a thread-per-request, where each thread handles a single request at a time

See www.baeldung.com/spring-webflux-concurrency
Comparing & Contrasting Spring WebMVC & WebFlux

• WebFlux is async

Callbacks are transparent to server code that uses Mono & Flux reactive types

Non-blocking I/O that leverages multiple cores & handles large # of connections
Comparing & Contrasting Spring WebMVC & WebFlux

- WebFlux is async
- It uses a completely non-blocking environment that can achieve higher concurrency with better resource utilization

See www.baeldung.com/spring-webflux-concurrency
Accessing Mono & Flux Endpoints Seamlessly
Accessing Mono & Flux Endpoints Seamlessly

- WebFlux Mono/Flux endpoints exchange HTTP requests/responses

See docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html
Accessing Mono & Flux Endpoints Seamlessly

- WebFlux Mono/Flux endpoints exchange HTTP requests/responses
- WebClient or RestTemplate can send/receive HTTP requests/responses to/from reactive endpoints

```java
Flux<Airport> airports = webClient.get()
    .uri(baseUrl + AIRPORT + "/*" + AIRPORTS)
    .retrieve()
    .bodyToFlux(Airport.class);
```

```java
Airport[] airports = restTemplate
    .getForEntity(baseUrl + AIRPORT + /* AIRPORTS,
    Airport[].class)
    .getBody();
```

See [www.baeldung.com/spring-webclient-resttemplate](http://www.baeldung.com/spring-webclient-resttemplate)
Accessing Mono & Flux Endpoints Seamlessly

- WebFlux Mono/Flux endpoints exchange HTTP requests/responses
- WebClient or RestTemplate can send/receive HTTP requests/responses to/from reactive endpoints

```java
Flux<Airport> airports = webClient
    .get()
    .uri(baseUrl + AIRPORT + "/" + AIRPORTS)
    .retrieve()
    .bodyToFlux(Airport.class);

Airport[] airports = restTemplate
    .getForEntity(baseUrl + AIRPORT + "/" + AIRPORTS,
                   Airport[].class)
    .getBody();
```

See [flights-microservices/-/blob/master/src/main/java/server/flight/FlightService.java](flights-microservices/-/blob/master/src/main/java/server/flight/FlightService.java)
Accessing Mono & Flux Endpoints Seamlessly

- WebFlux Mono/Flux endpoints exchange HTTP requests/responses
- WebClient or RestTemplate can send/receive HTTP requests/responses to/from reactive endpoints

```java
Airport[] airports = restTemplate
    .getForEntity(baseUrl + AIRPORT + "/" + AIRPORTS, Airport[].class)
    .getBody();
```

RestTemplate treats reactive types synchronously from the perspective of a client

However, no changes are required on the (reactive) server side
Accessing Mono & Flux Endpoints Seamlessly

- WebFlux Mono/Flux endpoints exchange HTTP requests/responses
- WebClient or RestTemplate can send/receive HTTP requests/responses to/from reactive endpoints

```java
Airport[] airports = restTemplate
    .getForEntity(baseUrl + AIRPORT + "/" + AIRPORTS,
                  Airport[].class)
    .getBody();

Flux<Airports> Flux.fromIterable(airports != null ? List.of(airports) : Collections.emptyList());
```

Easy to convert back to reactive types
Accessing Mono & Flux Endpoints Seamlessly

• WebFlux Mono/Flux endpoints exchange HTTP requests/responses
• WebClient or RestTemplate can send/receive HTTP requests/responses to/from reactive endpoints

```java
Flux<Airport> airports = webClient
    .get()
    .uri(baseUrl + AIRPORT + "/" + AIRPORTS)
    .retrieve()
    .bodyToFlux(Airport.class);
```

/WebClient leverages reactive types more effectively since responses are emitted as soon as they are available/

See [www.baeldung.com/spring-webflux-concurrency](http://www.baeldung.com/spring-webflux-concurrency)
Accessing Mono & Flux Endpoints Seamlessly

- WebFlux Mono/Flux endpoints exchange HTTP requests/responses
- WebClient or RestTemplate can send/receive HTTP requests/responses to/from reactive endpoints

```java
Flux<Airport> airports = webClient
    .get()
    .uri(baseUrl + AIRPORT + "/" + AIRPORTS)
    .retrieve()
    .bodyToFlux(Airport.class);
```

WebClient also enables end-to-end asynchrony

An HTTP request is not sent until subscribe() is called (& runs in thread pool)
Accessing Mono & Flux Endpoints Seamlessly

- WebFlux Mono/Flux endpoints exchange HTTP requests/responses
- WebClient or RestTemplate can send/receive HTTP requests/responses to/from reactive endpoints
- HTTP interface can also be used in Spring 6 & beyond in lieu of WebClient or RestTemplate

```java
Flux<Airport> mAsyncAirports = mAsyncAirportAPI.getAirports();
List<Airport> mSyncAirports = mSyncAirportAPI.getAirports();
```

See [www.baeldung.com/spring-6-http-interface](http://www.baeldung.com/spring-6-http-interface)
Accessing Mono & Flux Endpoints Seamlessly

- Jackson encoding/decoding is similar for reactive WebFlux Mono/Flux types or traditional WebMVC Ref Types/List types

```
GET flighthost:8081/airports

[
  {
    "airportCode": "ALB",
    "airportName": "Albany, NY"
  },
  {
    "airportCode": "AMA",
    "airportName": "Amarillo, TX"
  },
  {
    "airportCode": "ATL",
    "airportName": "Atlanta, GA"
  }, ...
```
Accessing Mono & Flux Endpoints Seamlessly

- JsonSerializer encoding/decoding is similar for reactive WebFlux Mono/Flux types or traditional WebMVC RefTypes/List types
- Tools like Postman can work seamlessly with either

See www.postman.com
End of Comparing & Contrasting Spring WebMVC & WebFlux