⚡Spring WebFlux Basics

🌳Introduction to Spring WebFlux

🌀Introduction

Spring WebFlux is a part of the **Spring 5 framework**, designed for building **reactive, non-blocking, and asynchronous web applications**. Unlike traditional Spring MVC (which is servlet-based and blocking), WebFlux leverages the **Reactive Streams API** to handle concurrency and scalability efficiently.

- Built on **Project Reactor** (Mono & Flux types).
- Provides **Reactive programming model**.
- Works with **Netty**, **Undertow**, or traditional servlet containers.
- Ideal for applications handling **high concurrency** with minimal resources.

🦙Subtopics of Spring WebFlux

1. 📖What is Spring WebFlux?
2. 💣Why WebFlux over Spring MVC?
3. ⚙️WebFlux Architecture
4. 🕐Reactive Types: Mono & Flux
5. 🧰Key Annotations in WebFlux
6. 🥣Functional vs. Annotation-based Endpoints
7. 🛠️Example: Simple WebFlux Application
8. ⚖️Advantages of WebFlux
9. 🕯️ Common Interview Questions on WebFlux

🐞1. What is Spring WebFlux?

Spring WebFlux is a **reactive programming framework** that allows building **asynchronous, non-blocking web applications**.

- Based on **Reactive Streams** specification.
- Uses **Publisher-Subscriber model**.
- Supports **two programming models**:
- Annotation-based (`@RestController`, `@GetMapping`)
- Functional endpoints (`RouterFunction`, `HandlerFunction`).

🌗 2. Why WebFlux over Spring MVC?

Feature	Spring MVC (Blocking)	Spring WebFlux (Non-blocking)
Concurrency Model	One thread per request	Event-loop model
Suitable For	Traditional apps	High-concurrency apps (chat, streaming)
Underlying Server	Servlet API (Tomcat)	Netty, Undertow, or servlet containers

🔔 WebFlux shines when dealing with **real-time data streams** and **microservices**.

---

3. WebFlux Architecture

- **Publisher** → Produces data stream.
- **Subscriber** → Consumes data stream.
- **Subscription** → Connects Publisher & Subscriber.
- **Backpressure** → Prevents overwhelming the consumer.

Spring WebFlux uses **Reactor (Mono & Flux)** to implement this.

🪻 4. Reactive Types: Mono & Flux

- **Mono** → Represents 0 or 1 element.
- **Flux** → Represents 0..N elements (stream of data).

```
@GetMapping("/mono")
public Mono<String> getMono() {
    return Mono.just("Hello from Mono");
}

@GetMapping("/flux")
public Flux<String> getFlux() {
    return Flux.just("A", "B", "C").delayElements(Duration.ofSeconds(1));
}
```

🔔 `Mono` is for single responses, `Flux` is for multiple streaming responses.

🍂 5. Key Annotations in WebFlux

- `@RestController` → Defines reactive REST controller.

- `@GetMapping`, `@PostMapping`, etc. → Handle HTTP requests.
- `@RequestBody` → Maps JSON to Java object.
- `@ResponseBody` → Converts Java object to JSON.
- `@EnableWebFlux` → Enables WebFlux configuration.

---

6. Functional vs. Annotation-based Endpoints

Annotation-based (similar to Spring MVC):

```
@RestController
class HelloController {
    @GetMapping("/hello")
    public Mono<String> sayHello() {
        return Mono.just("Hello WebFlux!");
    }
}
```

Functional style:

```
@Configuration
public class RouterConfig {
    @Bean
    public RouterFunction<ServerResponse> route(HelloHandler handler) {
        return RouterFunctions.route()
                .GET("/hello", handler::hello)
                .build();
    }
}

@Component
class HelloHandler {
    public Mono<ServerResponse> hello(ServerRequest request) {
        return ServerResponse.ok().body(Mono.just("Hello WebFlux Functional!"),
String.class);
    }
}
```

🔔Functional style is **lightweight** and often used in microservices.

🦠7. Example: Simple WebFlux Application

```
@SpringBootApplication
public class WebFluxDemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(WebFluxDemoApplication.class, args);
    }
}

@RestController
class UserController {
    @GetMapping("/users")
    public Flux<String> getUsers() {
        return Flux.just("Alice", "Bob",
"Charlie").delayElements(Duration.ofSeconds(1));
    }
}
```

Output (streamed with delay):

```
Alice
Bob
Charlie
```

🐒8. Advantages of WebFlux

- 🔗Non-blocking & asynchronous
- 🔗Handles high concurrency with fewer threads
- 🔗Streaming support with backpressure
- 🔗Runs on Netty, Undertow, or servlet containers
- 🔗Flexible: supports both annotation & functional models

🦦9. Common Interview Questions on WebFlux

1. **What is the difference between Spring MVC and WebFlux?**

2. MVC is blocking, WebFlux is non-blocking.

3. **What are Mono and Flux in WebFlux?**

4. Mono: 0/1 element, Flux: 0..N elements.

5. **Can WebFlux run on Tomcat?**

6. Yes, but Netty/Undertow are preferred.

7. **What is backpressure in Reactive Streams?**

8. Mechanism to prevent overwhelming the subscriber.

9. **When to use WebFlux instead of Spring MVC?**

10. Use WebFlux for high-concurrency apps (chat, IoT, streaming).

11. **Does WebFlux replace Spring MVC?**

12. No, both coexist. Choose based on use case.

🐻‍❄️Conclusion

Spring WebFlux is the **future-ready, reactive alternative** to traditional Spring MVC. By leveraging **Project Reactor (Mono, Flux)** and **Reactive Streams**, it enables developers to build **scalable, non-blocking applications** for modern cloud-native and microservices environments.
