1. What is Reactive Programming?

Reactive programming is a programming paradigm that deals with asynchronous data streams and the specific propagation of change, which means it implements modifications to the execution environment in a certain order.

The primary benefits of reactive programming are, better utilization of computing resources on multicore and multi-CPU hardware and better performance by truncating serialization.

2. What is Spring Webflux?

Spring WebFlux is part of Spring 5, it is a parallel version of Spring MVC and supports fully non-blocking reactive streams. It supports the backpressure concept and uses Netty as the inbuilt server to run reactive applications.

3. Is Spring WebClient thread safe?

Yes, WebClient is thread-safe since it is immutable.

4. Is Spring RestTemplate deprecated?

RestTemplate allows you to consume Rest resources in a synchronous manner, which means it will block the thread before it receives a response. RestTemplate has been deprecated since Spring 5.

5. What are the benefits of using Spring Webflux?

The spring-webflux is not necessarily faster but it is better in dealing with the issues of scalability and efficient use of hardware resources. It is also better at dealing with the issues of latency.

6. What is bodyToMono?

The method bodyToMono() is extracts the body to a Mono instance. The method Mono.block() subscribes to this Mono instance and blocks until the response is received while Mono.subscribe() method is non-blocking.

7. Difference between Mono and Flux.

A Mono, which can either return zero or one result before completing,

And a Flux, which can return zero to many, possibly infinite, results before completing.

8. How to handle errors in Spring WebFlux?

Handle Errors With,

* **onErrorReturn** (), to return a static default value whenever an error occurs.
* **onErrorResume**(), to compute a dynamic fallback value, (or) execute an alternative path with a fallback method, (or) Catch, wrap and re-throw an error.
* **Global Level**by customizing the Global Error Response Attributes and implement the Global Error Handler.

9. Difference between Spring MVC @async and Spring Webflux?

Spring Async I/O model during its communication with the client is blocking while Spring Webflux doesn't block.

Spring WebFlux supports more Web/App servers such as Netty, Undertow.

Processing the request body in Spring Async is blocking while it is non-blocking with Spring Webflux.

10. What is Reactor netty?

Reactor Netty is an asynchronous event-driven network application framework. It provides non-blocking and backpressure-ready TCP, HTTP, and UDP clients and servers. As the name implies, it's based on the Netty framework.

11. What are router functions?

RouterFunction is a functional alternative to the @RequestMapping and @Controller annotation style used in standard Spring MVC.

We can use it to route requests to the handler functions.

|  |  |
| --- | --- |
| 1  2  3  4  5 | **@Bean**  **public** RouterFunction<ServerResponse> **listEmployees**(EmployeeService es) {  **return** **route**().GET("/employee", req -> ok().body(es.findAll()))  .build();  } |

12. What does Mono.defer() do?

Mono. defer(monoSupplier) lets you provide the whole expression that supplies the resulting Mono instance. The evaluation of this expression is deferred until somebody subscribes. Inside of this expression you can additionally use control structures like Mono.

When you run Mono.just() it creates immediately an Observable(Mono)and reuses it but when you use defer it doesn't create it immediately it creates a new Observable in every subscribe.

13. When should we choose Reactor over RxJava?

RxJava support project which is based on JDK8- and Project Reactor supports JDK 8+. RxJava has too many problems which can cause Out of Memory if you can't use it well. It is preferred to use Reactor for JDK8+ projects.

14. How to handle an exception in a Spring Webflux based application?

Different ways to handle exceptions.

* onErrorReturn,
* onErrorResume,
* onErrorMap,
* and Handling Errors at a Global Level.

15. What is bodyToFlux?

Similar to bodyToMono, however here it is used to to retrieve a collection resource of type Flux from endpoint /stocks.

Flux<Stock> stockFluxObj = client.get()

.uri("/stocks")

.retrieve()

.bodyToFlux(Stock.class);

stockFluxObj.subscribe(System.out::println);

16. Disadvantages of Using Reactive Streams.

Troubleshooting a Reactive application is difficult compared to the regular applications.

There is an extra learning curve when implementing reactive applications.

There is limited support for reactive data stores since traditional relational data stores have yet to embrace the reactive paradigm.

17. Difference between the Web client and Web Test Client?

|  |  |
| --- | --- |
| **WebClient** | **WebTestClient** |
| Web client acts as a reactive client who performs non-blocking HTTP requests. | Webtestclient also acts as a reactive client that can be used in tests.. |
| It can handle reactive streams with backpressure. | It can bind directly to WebFlux application by applying mock request and response objects. |
| It can take advantage of JAVA 8 Lambdas. | It can connect to any server over an HTTP connection. |

18. Can we use SpringMvc and webflux together?

No. SpringMVC cannot be run on Netty.

19. What are the main components of Spring Webflux?

There are 2 main components of Spring Webflux: **RouterFunction** and the **HandlerFunction**. The RouterFunction is responsible for mapping incoming requests to the appropriate HandlerFunction. The HandlerFunction is responsible for handling the request and returning a response.

20. What is the Flux API?

The Flux API is a reactive programming library for Java that allows developers to easily build asynchronous applications. The library is based on the Reactive Streams specification, and provides a wide variety of operators that can be used to manipulate data streams.

21. What is a Publisher? Explain its main interfaces.

A Publisher is a reactive component that emits a stream of data. The two main interfaces are Publisher and Subscriber.

22. Explain the difference between hot publishers and cold publishers.

A hot publisher is a publisher that emits data even if there is no Subscriber subscribed to it. A cold publisher is a publisher that only emits data when there is at least one Subscriber subscribed to it.

23. Explain throttling.

Throttling limits the amount of data that can be transferred between two systems in a given period of time. This is often done to prevent one system from overloading another system with too much data, or to prevent one user from consuming too much of a shared resource.

**Spring WebFlux Interview Questions and Answers**

Here are 20 commonly asked Spring WebFlux interview questions and answers to prepare you for your interview:

1. What is WebFlux?

WebFlux is a reactive programming model for building web applications. It is built on top of the Reactive Streams API and provides a way to build event-driven, non-blocking applications.

2. How does Spring Webflux differ from Spring MVC?

Spring Webflux is a non-blocking, reactive framework for building web applications. This means that instead of the traditional request/response model, Webflux uses an event-driven model where the application reacts to events as they occur. This can lead to better performance and scalability, as well as a more responsive user experience. Spring MVC is the traditional, blocking framework for building web applications.

3. Can you explain how to create a web service using Spring Boot Webflux?

You can create a web service using Spring Boot Webflux by following these steps:  
1. Create a Spring Boot project  
2. Add the WebFlux dependency  
3. Create a controller class and annotate it with @RestController  
4. Create a request mapping method and annotate it with @GetMapping  
5. Run the application

4. What are the main components of Spring Webflux?

The main components of Spring Webflux are the RouterFunction and the HandlerFunction. The RouterFunction is responsible for mapping incoming requests to the appropriate HandlerFunction. The HandlerFunction is responsible for handling the request and returning a response.

5. Is it possible to use reactive programming in Spring Webflux? If yes, then how?

Yes, it is possible to use reactive programming in Spring Webflux. This can be accomplished by using the reactive programming support that is built into the Spring Framework. This support allows for the creation of reactive components that can be used in conjunction with Spring Webflux to create reactive applications.

6. In which situations would I want to use reactive programming over traditional techniques like multithreading and promises?

Reactive programming can provide benefits over traditional techniques in situations where you need to process a large number of small requests quickly, or where you need to be able to handle a variable number of requests. Additionally, reactive programming can make it easier to reason about your code, since you can think about it in terms of data flows instead of individual threads of execution.

7. What’s the difference between blocking vs non-blocking calls? Which one do you think is better and why?

Blocking calls are those where the thread making the call is blocked until the call returns. Non-blocking calls are those where the thread making the call is not blocked, and can continue to do other work while the call is being processed. I think that non-blocking calls are better because they allow for better utilization of resources.

8. What is event looping? Why is it important for asynchronous processing?

Event looping is a process whereby a program continually checks for new events or tasks and then processes them as they come in. This is important for asynchronous processing because it allows the program to continue running even while waiting for new tasks to come in, which can improve efficiency and performance.

9. When should I prefer an async/await approach over a promise approach when writing code in JavaScript?

There is no definitive answer to this question, as it depends on the specific situation and what you are trying to achieve. However, in general, an async/await approach may be preferable if you are working with code that is already asynchronous or if you need to perform multiple asynchronous operations in a row. A promise approach may be more appropriate if you are working with code that is not asynchronous or if you only need to perform a single asynchronous operation.

10. What is the purpose of the Future class in Java?

The Future class is used to represent the result of an asynchronous computation. This is useful in a Spring WebFlux application, where you may need to perform a long-running task in the background and then return the result to the caller. By using a Future, you can start the computation immediately and then wait for it to finish before returning the result.

11. What do you understand about executing tasks in parallel?

When you execute a task in parallel, you are essentially running multiple tasks at the same time. This can be useful when you want to speed up the execution of a task by running it on multiple processors or cores. It can also be useful for running tasks that are independent of each other, so that one task does not have to wait for the other to finish before it can start.

12. Can you give me some examples of real-world applications that can benefit from the usage of reactive programming?

Real-world applications that can benefit from reactive programming include those that deal with high volumes of data or those that need to be able to handle a large number of concurrent users. Reactive programming can help to make these applications more responsive and scalable.

13. What is the best way to handle errors while performing asynchronous operations?

The best way to handle errors while performing asynchronous operations is to use the onErrorResume method. This will allow you to resume the operation in the event of an error, which can help to prevent the entire operation from being aborted.

14. What is your opinion on the performance tradeoffs associated with using reactive programming?

Reactive programming can provide a significant performance boost over traditional programming techniques, but it comes at the cost of increased complexity. If you are willing to trade off some simplicity for the sake of performance, then reactive programming may be the right choice for your project.

15. What is Reactor? What are its main components?

Reactor is a library that helps you build reactive applications in Java. It is based on the Reactive Streams specification. The main components of Reactor are the Publisher and the Subscriber. The Publisher is responsible for emitting data, and the Subscriber is responsible for consuming that data.

16. What is the Flux API?

The Flux API is a reactive programming library for Java that allows developers to easily build asynchronous applications. The library is based on the Reactive Streams specification, and provides a wide variety of operators that can be used to manipulate data streams.

17. What is a Publisher? What are its two main interfaces?

A Publisher is a reactive component that emits a stream of data. The two main interfaces are Publisher and Subscriber.

18. What are hot publishers and cold publishers?

A hot publisher is a publisher that emits data even if there is no Subscriber subscribed to it. A cold publisher is a publisher that only emits data when there is at least one Subscriber subscribed to it.

19. What are backpressure operators? What are their advantages?

Backpressure operators are a type of operator specifically designed for use with reactive streams. They help to ensure that a stream does not become overwhelmed by too much data, and can instead process data at a rate that it can comfortably handle. This can help to avoid issues such as buffering and blocking, and can improve overall performance.

20. What is throttling?

Throttling is a process of limiting the amount of data that can be transferred between two systems in a given period of time. This is often done to prevent one system from overloading another system with too much data, or to prevent one user from consuming too much of a shared resource.

Top 25 Spring WebFlux Interview Questions and Answers

Explore our comprehensive guide on Spring WebFlux interview questions and answers to enhance your knowledge and prepare effectively for your upcoming interviews in the field of web development.

[](https://interviewprep.org/users/8535/profile/interviewprep-it-career-coach)

[InterviewPrep IT Career Coach](https://interviewprep.org/users/8535/profile/interviewprep-it-career-coach)

Published Sep 8, 2023

Spring WebFlux, a part of the larger Spring Framework 5.0, has emerged as an innovative solution for creating non-blocking, reactive web applications. This open-source framework is designed to leverage asynchronous I/O with a functional programming model, making it ideal for handling streaming and event-driven workloads.

In contrast to traditional servlet-based blocking I/O, Spring WebFlux adopts a non-blocking approach which allows it to handle large numbers of concurrent connections with minimal threads, providing significant scalability advantages. It’s particularly well-suited for high-performance systems where latency can’t be compromised.

This article presents an extensive collection of interview questions related to Spring WebFlux. Whether you’re a novice developer or seasoned professional, these questions will help you delve deeper into the world of reactive programming. From understanding the basics of Spring WebFlux to exploring its integration with other technologies in the Spring ecosystem, this guide aims to equip you with knowledge that will give you an edge in your technical interviews.

1. Can you explain what Spring WebFlux is and how it differs from traditional Spring Web MVC?

Spring WebFlux is a non-blocking, reactive web framework part of Spring 5.0+ for building scalable applications. Unlike traditional Spring MVC, it operates on event-loop model rather than servlet-based. It’s designed to handle large number of concurrent connections with minimal threads, improving scalability and resource utilization.

WebFlux supports both annotated controllers and functional endpoints. The main difference from MVC lies in the core architecture: MVC uses Servlet API and runs on servlet containers while WebFlux doesn’t rely on Servlet API, enabling it to run on servers like Netty.

In terms of data handling, MVC blocks until data is available or processed whereas WebFlux employs backpressure concept from Reactive Streams specification, allowing it to deal with streams of data efficiently without blocking.

2. How does Spring WebFlux implement the Reactive Streams specification?

Spring WebFlux implements the Reactive Streams specification through its core library, Project Reactor. It provides two main types: Flux and Mono, representing 0-N and 0-1 elements respectively. These types are used to represent asynchronous sequences of data. Spring WebFlux uses these types in its API for handling HTTP requests and responses.

The framework also supports backpressure, a key feature of Reactive Streams, allowing consumers to control how much data they receive. This is achieved by using Project Reactor’s operators that can request specific amounts of data or apply rate limiting strategies.

Furthermore, Spring WebFlux leverages non-blocking I/O operations provided by Netty, an asynchronous networking library. This allows it to handle many concurrent connections with minimal threads, improving scalability.

3. Can you describe a situation where you’d prefer to use WebFlux over MVC?

WebFlux is preferred over MVC in scenarios where high concurrency and non-blocking operations are required. For instance, when developing a service that interacts with multiple external APIs or databases concurrently, WebFlux’s non-blocking nature allows for better resource utilization as it doesn’t tie up server threads waiting for responses. This leads to improved scalability under heavy load. Additionally, if the application needs to handle real-time data streams, WebFlux’s support for reactive programming makes it an ideal choice.

4. How does Spring WebFlux handle backpressure and why is it important?

Spring WebFlux handles backpressure through its non-blocking, reactive programming model. It uses Project Reactor’s Flux and Mono types which support backpressure by default. When a data source produces items faster than the consumer can handle, backpressure comes into play. The consumer signals to the producer how many items it can process, preventing system overload.

Backpressure is crucial in maintaining system stability. Without it, fast producers could overwhelm slower consumers leading to resource exhaustion. This could result in application failure or degraded performance. By managing the rate of data flow, Spring WebFlux ensures optimal utilization of resources, enhancing overall system resilience and responsiveness.

5. What are the key differences between an imperative and a reactive programming model?

Imperative programming is a traditional model where operations are executed sequentially, blocking the thread until completion. It’s efficient for simple tasks but struggles with high-load systems due to limited threads.

Reactive programming, on the other hand, is an asynchronous model that handles multiple requests concurrently without blocking threads. This non-blocking nature makes it ideal for high-load systems as it can handle more requests with fewer resources.

In Spring WebFlux, reactive programming is implemented using Project Reactor which provides Flux (0-N items) and Mono (0-1 item) for handling streams of data asynchronously. The main advantage here is backpressure – the ability of consumers to control the rate of data they receive from producers, preventing overloading.

6. Can you explain how non-blocking IO works in Spring WebFlux?

Spring WebFlux, a reactive-stack web framework, utilizes non-blocking IO to handle concurrent requests without blocking threads. It achieves this through the use of Project Reactor and its Flux and Mono types which represent multiple and single results respectively.

In traditional servlet-based applications, each request is handled by a dedicated thread from a limited pool, potentially causing resource exhaustion under heavy load. However, in Spring WebFlux, when a request arrives, it’s not assigned to a specific thread. Instead, tasks are broken down into smaller parts that can be executed upon data availability or event occurrence.

This approach allows for more efficient utilization of resources as threads aren’t blocked waiting for IO operations to complete. The server can handle many more concurrent connections with fewer threads, improving scalability and resilience.

7. How does Spring WebFlux benefit from a non-blocking web stack compared to traditional servlet containers?

Spring WebFlux, a non-blocking web stack, offers several advantages over traditional servlet containers. It enables efficient use of resources by not blocking threads while waiting for network input/output operations to complete. This is particularly beneficial in high-load scenarios where many concurrent requests need to be handled. The reactive programming model it uses allows developers to write code that can handle backpressure, ensuring system stability under heavy load. Furthermore, Spring WebFlux supports both HTTP/2 and WebSocket protocols, providing more flexibility in communication between client and server.

8. What is the role of the Project Reactor in Spring WebFlux?

Project Reactor, a reactive programming library for Java, plays an integral role in Spring WebFlux. It provides the foundation for non-blocking and backpressure-ready data pipelines. The two main types of publishers it offers are Mono (0 or 1 result) and Flux (0 to N results), which are used extensively in WebFlux for handling HTTP requests asynchronously. This allows applications to handle more concurrent users with fewer threads, improving scalability.

9. Can you explain the difference between Mono and Flux in Project Reactor?

Mono and Flux are core types in Project Reactor. Mono is used to represent a sequence of 0 or 1 item, while Flux represents a sequence of 0 to N items. Essentially, Mono would be used when we expect a single result, whereas Flux should be used for multiple or no results. Both support transformation and action operations. However, the key difference lies in their use cases: Mono is ideal for scenarios where you’re dealing with asynchronous computations that yield a single result (like database queries), while Flux is better suited for handling streams of data (like processing live updates).

10. What types of databases and data repositories can be used effectively with Spring WebFlux?

Spring WebFlux can effectively work with both SQL and NoSQL databases. For SQL, it supports R2DBC (Reactive Relational Database Connectivity), which provides non-blocking database access. It’s compatible with PostgreSQL, MySQL, Microsoft SQL Server, and H2. For NoSQL, Spring Data offers reactive support for MongoDB, Cassandra, Couchbase, and Redis. These repositories provide a way to perform CRUD operations in a non-blocking manner, making them suitable for use with Spring WebFlux.

11. How would you handle exception propagation in Spring WebFlux?

In Spring WebFlux, exception propagation is handled using the onError\* operators provided by Project Reactor. These operators allow you to define a fallback mechanism when an error occurs during the processing of a Flux or Mono stream.

For instance, if we have a method that returns a Mono and it throws an exception, we can handle this with onErrorReturn operator:

**public** Mono<**String**> findUser(**String** id) {

**return** userRepository.findById(id)

.onErrorReturn("Default User");

}

Here, if findById throws an exception, “Default User” will be returned instead.

Another way to propagate exceptions is through @ExceptionHandler in @ControllerAdvice classes. This allows for centralized exception handling across all @RequestMapping methods. For example:

@ControllerAdvice

**public** **class** GlobalExceptionHandler {

@ExceptionHandler(RuntimeException.class)

**public** ResponseEntity<**String**> handleRuntimeException(RuntimeException ex) {

**return** ResponseEntity.status(HttpStatus.BAD\_REQUEST).body(ex.getMessage());

}

}

12. Can you discuss the thread model in Spring WebFlux and how it differs from a traditional servlet model?

Spring WebFlux employs a non-blocking, event-driven model for handling concurrency and scales with fewer threads. It uses Reactor to provide asynchronous processing and backpressure-ready libraries. Unlike the traditional servlet model which utilizes one thread per request, Spring WebFlux can handle multiple requests within a single thread due to its non-blocking nature. This results in efficient resource utilization as it doesn’t require creating new threads for each incoming request. The servlet model’s blocking I/O operations could lead to thread starvation if all threads are busy, while Spring WebFlux avoids this issue by freeing up the thread when waiting for data, allowing it to serve other requests.

13. How do you test reactive endpoints in Spring WebFlux?

To test reactive endpoints in Spring WebFlux, use the WebTestClient. This non-blocking client can be used for testing web handlers and controllers. It binds to a RouterFunction or any arbitrary WebHandler function, allowing you to test your routes directly without requiring an actual server.

For instance, if you have a router function:

RouterFunction<?> route = ...

WebTestClient

.bindToRouterFunction(route)

.build()

.get().uri("/path")

.exchange()

.expectStatus().isOk()

.expectBody(String.class).isEqualTo("expected response");

In this example, we’re binding our WebTestClient to the router function, then making a GET request to “/path”. We expect a status of OK (200) and a body that equals “expected response”.

14. How would you secure a Spring WebFlux application?

To secure a Spring WebFlux application, you would use Spring Security’s reactive features. This includes authentication and authorization mechanisms that are non-blocking and support back pressure. You can configure security by creating a SecurityWebFilterChain bean in your configuration class.

For example:

@Bean

**public** SecurityWebFilterChain springSecurityFilterChain(ServerHttpSecurity http) {

http.authorizeExchange()

.anyExchange().authenticated()

.and().httpBasic().and().formLogin();

**return** http.build();

}

This code ensures all requests require authentication and supports both HTTP Basic and form-based login.

You may also want to enable CSRF protection if the application is serving browser clients. For JWT token based security, you need to create custom authentication and authorization filters. Remember to always keep sensitive data like passwords and tokens secure using encryption or hashing techniques.

15. How would you implement server-sent events in Spring WebFlux?

To implement server-sent events in Spring WebFlux, you would use the ServerSentEvent class. First, create a REST controller and define an endpoint that produces “text/event-stream”. This is the media type for server-sent events. Then, return a Flux from this endpoint.

Here’s a simple example:

@RestController

**public** **class** SseController {

@GetMapping(value = "/sse", produces = MediaType.TEXT\_EVENT\_STREAM\_VALUE)

**public** Flux<ServerSentEvent<**String**>> getEvents() {

**return** Flux.interval(Duration.ofSeconds(1))

.map(sequence -> ServerSentEvent.<**String**> builder()

.id(**String**.valueOf(sequence))

.event("periodic-event")

.data("SSE - " + LocalTime.now().toString())

.build());

}

}

In this code, we’re creating a new event every second with the current time as data.

16. How does Spring WebFlux support CORS policy?

Spring WebFlux supports CORS policy through its built-in mechanisms. It provides @CrossOrigin annotation at both class and method level to handle cross-origin requests. This annotation allows you to define which origins, HTTP methods, headers are allowed, and whether credentials are supported.

For global configuration, Spring WebFlux uses the CorsRegistry in WebFluxConfigurer interface. You can register a CorsConfiguration object with specific paths to apply CORS rules globally.

In addition, it also offers functional style routing where you can use RouterFunction to set up CORS. The RouterFunctions.cors() method is used here to wrap all router functions with a default CORS configuration.

17. Can you explain how to achieve concurrency in Spring WebFlux?

Spring WebFlux achieves concurrency through non-blocking, asynchronous programming. It uses Reactor library which provides Mono and Flux API types representing single and multiple values over time respectively. These APIs are lazy, only executing computations upon subscription.

In a traditional Spring MVC application, each request is handled by a dedicated thread until the response is ready. This can lead to inefficient use of resources when dealing with I/O bound tasks such as database or network calls.

However, in Spring WebFlux, threads aren’t blocked. When an operation is dependent on I/O, control is released back to the event loop allowing other tasks to be processed. Once the I/O operation completes, it gets scheduled back onto the event loop for further processing.

This model allows handling many concurrent requests with fewer threads, leading to efficient resource utilization. The key here is that the underlying infrastructure (like databases, web services) must also support non-blocking operations to fully leverage this model.

18. Can you illustrate with examples the difference between the cold and hot publishers in Project Reactor?

Cold publishers generate data for each subscription. For instance, if we have a Flux that reads from a file, every time we subscribe to it, the reading process starts anew.

Example:

Flux<String> cold = Flux.just(readFile("file.txt"));

cold.subscribe(); // Reads the file

cold.subscribe(); // Reads the file again

Hot publishers share their data with all subscribers simultaneously. If a subscriber joins late, it might miss some items. An example is a stock ticker where new subscribers only get updates from the point of joining.

Example:

DirectProcessor<String> hot = DirectProcessor.create();

hot.onNext("stock update");

hot.subscribe(System.out::println); // Won't receive "stock update"

hot.onNext("new stock update"); // Will print "new stock update"

19. How do you deal with backpressure in a Spring WebFlux application?

In Spring WebFlux, backpressure is handled through the implementation of Project Reactor’s Flux and Mono types. These are reactive streams that support backpressure by default. When a client cannot keep up with the data rate, it sends a request to the server specifying how much data it can handle. This is known as demand. The server then respects this demand and only sends the specified amount of data. If the client needs more data, it sends another request. This mechanism prevents overwhelming the client with too much data at once.

20. How do you handle form validation and error handling in Spring WebFlux?

Spring WebFlux handles form validation and error handling through the use of Data Binding and Validation API. The @Valid annotation is used to trigger validation for a model attribute, typically in a request handler method. Errors can be checked using BindingResult which follows the command object that is validated.

For example:

@PostMapping("/save")

public Mono<String> save(@Valid Person person, BindingResult result) {

**if** (result.hasErrors()) {

**return** Mono.just("form");

}

// handle success case

}

In this code snippet, an incoming POST request with data bound to a ‘Person’ object triggers validation. If there are errors, it returns a view named “form”. Otherwise, it proceeds with the successful scenario.

Error messages can be customized by defining message source bean in configuration class or properties file. For field-specific errors, we use FieldError’s getField and getDefaultMessage methods.

21. Can you explain how to handle Stream and Batch processing in Spring WebFlux?

Spring WebFlux, a non-blocking web framework, handles stream and batch processing differently than traditional Spring MVC. For streaming data, Server-Sent Events (SSE) are used. SSE is an HTTP standard that allows real-time transfer of events from server to client. In Spring WebFlux, you can return a Flux where T is the type of object being streamed.

For example:

@GetMapping(value = "/stream", produces = MediaType.TEXT\_EVENT\_STREAM\_VALUE)

**public** Flux<**String**> streamData() {

**return** Flux.interval(Duration.ofSeconds(1)).map(sequence -> "Stream Value: " + sequence);

}

This code streams a new string every second.

Batch processing in Spring WebFlux uses backpressure concept. Backpressure controls how fast the data is consumed from a Flux or Mono. You can set the demand for number of elements to process at a time using

request(n)

 method on

Subscription

 object.

For example:

flux.subscribe(new Subscriber<**Integer**>() {

@Override

**public** **void** onSubscribe(Subscription s) {

s.request(10); // Process 10 elements at a time

}

});

22. How do you track and manage latency issues in Spring WebFlux?

Spring WebFlux, being non-blocking and reactive, inherently manages latency issues. However, to track these, Micrometer is a useful tool that integrates with Spring Boot Actuator. It provides dimensional metrics data which can be sent to various monitoring systems.

To manage latency, you can use backpressure strategies provided by Project Reactor. Backpressure allows consumers to signal producers about their readiness to process data, preventing system overload.

Another strategy is tuning the number of threads in Netty’s event loop group. This should align with the number of CPU cores for optimal performance.

Also, consider using WebClient instead of RestTemplate for HTTP requests as it operates on non-blocking I/O.

23. How do you handle sessions in a Spring WebFlux application?

In Spring WebFlux, sessions are handled using the ServerWebExchange object. This object provides a getSession() method that returns a Mono. The WebSession interface allows you to get and set attributes in the session. To use it, inject ServerWebExchange into your handler method or retrieve it from the routing context if you’re using the functional style of routing. You can then call exchange.getSession().doOnNext(session -> {session.getAttributes().put(“key”, “value”);}). Here’s an example:

**public** Mono<ServerResponse> handleRequest(ServerRequest request) {

**return** request.session()

.flatMap(session -> {

session.getAttributes().put("key", "value");

**return** ServerResponse.ok().build();

});

}

24. How does Spring WebFlux handle multipart file uploads?

Spring WebFlux handles multipart file uploads using the

Part

 interface and its subclass

FilePart

. The

@RequestPart

 annotation is used to bind parts of a multipart request, including files. When a multipart/form-data POST request is received, Spring WebFlux splits it into multiple parts. Each part can be accessed as a

Mono<Part>

 or

Flux<Part>

, depending on whether one or many are expected. For file parts specifically, they’re handled as

FilePart

 instances which provide methods for transferring the file to a local disk or streaming it to a

DataBuffer

. This non-blocking handling allows efficient processing of large files without consuming excessive resources.

25. Can you explain how to implement WebSockets with Spring WebFlux?

Spring WebFlux supports WebSocket protocol. To implement, first define a WebSocketHandler to handle messages. Use WebSocketSession’s receive() method for incoming messages and send() for outgoing.

Here is an example:

@Component

**public** **class** MyWebSocketHandler **implements** WebSocketHandler {

@Override

**public** Mono<Void> handle(WebSocketSession session) {

**return** session.send(

session.receive()

.map(msg -> session.textMessage("Echo: " + msg.getPayloadAsText()))

);

}

}

Next, register the handler with a WebSocketHandlerAdapter and map it to a URL using RouterFunction.

@Configuration

**public** **class** WebSocketConfig {

@Autowired

**private** MyWebSocketHandler myWebSocketHandler;

@Bean

**public** HandlerMapping handlerMapping() {

Map<**String**, WebSocketHandler> map = new HashMap<>();

map.put("/echo", myWebSocketHandler);

SimpleUrlHandlerMapping mapping = new SimpleUrlHandlerMapping();

mapping.setUrlMap(map);

mapping.setOrder(-1); // before annotated controllers

**return** mapping;

}

@Bean

**public** WebSocketHandlerAdapter handlerAdapter() {

**return** new WebSocketHandlerAdapter();

}

}

24 Spring WebFlux Interview Questions and Answers

Introduction:

As you prepare for your Spring WebFlux interview, whether you are an experienced professional or a fresher entering the programming world, it's essential to be well-versed in the common questions that might be thrown your way. This blog provides a comprehensive list of 24 Spring WebFlux interview questions and detailed answers to help you navigate through the interview process successfully. Whether you are familiar with reactive programming or just starting, these questions cover a range of topics to ensure you're well-prepared for any scenario.

Role and Responsibility of a Spring WebFlux Developer:

Spring WebFlux developers play a crucial role in building reactive, scalable, and non-blocking applications. They are responsible for designing and implementing features using the reactive programming paradigm, handling asynchronous data streams, and optimizing performance for high-concurrency scenarios. Understanding the intricacies of Spring WebFlux and its integration with other Spring components is vital for delivering robust and responsive applications.

Common Interview Question Answers Section:

1. What is Reactive Programming?

Reactive Programming is a programming paradigm that focuses on handling asynchronous data streams. It enables the development of applications that can react to changes in data in real-time, providing responsiveness and scalability.

**How to answer:** Explain the concept of reactive programming, mentioning the use of Observables and the importance of handling events asynchronously for better performance.

**Example Answer:** *"Reactive Programming is an approach where applications react to changes in data rather than actively polling for updates. In Spring WebFlux, this is achieved through the use of reactive types like Flux and Mono, allowing us to handle asynchronous data streams efficiently."*

2. What are the key components of Spring WebFlux?

Spring WebFlux consists of key components that enable the development of reactive applications. Understanding these components is essential for effective usage.

**How to answer:** Discuss the main components, such as DispatcherHandler, HandlerMapping, and WebClient, and explain their roles in the reactive application architecture.

**Example Answer:** *"The key components of Spring WebFlux include DispatcherHandler, which routes requests to the appropriate handlers, HandlerMapping, responsible for mapping requests to handler methods, and WebClient for making reactive HTTP requests. These components work together to support the reactive programming model."*

3. Explain the difference between Flux and Mono in Spring WebFlux.

Flux and Mono are fundamental types in Spring WebFlux for handling reactive streams. Understanding their differences is crucial for effective usage.

**How to answer:** Clearly define Flux as a representation of a reactive stream with multiple elements, while Mono represents a stream with at most one element. Emphasize the use cases where each is applicable.

**Example Answer:** *"Flux is used for handling multiple elements in a reactive stream, while Mono is suitable for scenarios where there's at most one element. For instance, Flux is ideal for handling multiple values from a database query, while Mono is suitable for a single result or a computation that might emit a single value."*

4. How does error handling work in Spring WebFlux?

Error handling is crucial in any application, and in Spring WebFlux, it's important to understand how to handle errors in a reactive environment.

**How to answer:** Explain the use of operators like `onErrorResume` and `onErrorReturn` in handling errors in reactive streams. Discuss the importance of a global error handler and the role of the `ExceptionHandler` class.

**Example Answer:** *"In Spring WebFlux, error handling is achieved using operators like `onErrorResume` to provide an alternative value or stream in case of an error. The `ExceptionHandler` class allows for global error handling. Additionally, `onErrorReturn` can be used to return a default value when an error occurs."*

5. How do you achieve method-level security in a Spring WebFlux application?

Securing methods in a Spring WebFlux application is essential for protecting sensitive operations. Understanding the method-level security features is key.

**How to answer:** Discuss the use of annotations like `@PreAuthorize` and `@Secured` to secure methods. Mention the role of `ReactiveSecurityContextHolder` in obtaining security information.

**Example Answer:** *"Method-level security in Spring WebFlux is achieved through annotations like `@PreAuthorize` and `@Secured`. These annotations allow us to specify access control rules. The `ReactiveSecurityContextHolder` can be used to obtain security context information within reactive methods."*

6. What is R2DBC, and how does it differ from JDBC in the context of Spring WebFlux?

R2DBC is a reactive relational database connectivity library, and understanding its differences from JDBC is essential for building reactive database interactions in Spring WebFlux.

**How to answer:** Explain that R2DBC is designed for reactive programming, allowing non-blocking database access, while JDBC follows a blocking approach. Discuss the benefits of R2DBC in handling concurrent requests in a reactive application.

**Example Answer:** *"R2DBC is a reactive alternative to JDBC, offering non-blocking database access. Unlike JDBC, which follows a blocking approach, R2DBC is designed for reactive programming. It allows for efficient handling of concurrent database requests in a reactive and non-blocking manner."*

7. What is the purpose of the `@RestController` annotation in Spring WebFlux?

The `@RestController` annotation plays a key role in defining controllers in a Spring WebFlux application, and understanding its purpose is fundamental.

**How to answer:** Explain that `@RestController` is a specialized version of `@Controller` that combines `@Controller` and `@ResponseBody`. It is used to define RESTful web services that directly return data rather than relying on view resolution.

**Example Answer:** *"The `@RestController` annotation in Spring WebFlux is used to define controllers for RESTful web services. It combines the functionality of `@Controller` and `@ResponseBody`, indicating that the return value of methods should be directly serialized into the response body, making it suitable for building APIs."*

8. How does Spring WebFlux handle blocking operations?

Understanding how Spring WebFlux deals with blocking operations is crucial for building efficient and responsive applications.

**How to answer:** Explain the role of schedulers in Spring WebFlux, such as the use of `Schedulers.elastic()` for offloading blocking operations to a separate thread pool, ensuring that the main event loop remains non-blocking.

**Example Answer:** *"Spring WebFlux handles blocking operations by using schedulers. The `Schedulers.elastic()` scheduler, for instance, allows us to offload blocking tasks to a separate thread pool. This ensures that the main event loop remains non-blocking, maintaining the reactivity of the application."*

9. What is the purpose of the `@EnableWebFlux` annotation?

The `@EnableWebFlux` annotation is crucial for configuring and enabling Spring WebFlux in a project. Understanding its purpose is key for setting up a reactive environment.

**How to answer:** Explain that `@EnableWebFlux` is used to enable the Spring WebFlux configuration in a project. It activates reactive features, such as annotated controllers and reactive repositories.

**Example Answer:** *"The `@EnableWebFlux` annotation is used to enable the configuration of Spring WebFlux in a project. It activates the necessary components for building reactive applications, including annotated controllers and reactive repositories."*

10. How does Spring WebFlux support WebSocket communication?

WebSocket communication is essential for real-time, bidirectional communication in web applications. Understanding how Spring WebFlux supports WebSocket is crucial for building interactive features.

**How to answer:** Explain that Spring WebFlux supports WebSocket communication through the `WebSocketHandler` interface and the use of `@MessageMapping` to handle incoming messages. Discuss the role of `WebSocketConfigurer` in configuring WebSocket support.

**Example Answer:** *"Spring WebFlux supports WebSocket communication through the `WebSocketHandler` interface, allowing bidirectional communication. We can use the `@MessageMapping` annotation to handle incoming messages. Additionally, the `WebSocketConfigurer` interface enables us to configure WebSocket support in our application."*

11. What is the role of the `@Autowired` annotation in Spring WebFlux?

The `@Autowired` annotation is fundamental in the Spring framework for dependency injection. Understanding its role in Spring WebFlux is crucial for managing and injecting dependencies.

**How to answer:** Explain that `@Autowired` is used for automatic dependency injection in Spring WebFlux. It injects dependencies into beans, reducing the need for manual configuration.

**Example Answer:** *"In Spring WebFlux, the `@Autowired` annotation is used for automatic dependency injection. It allows us to inject dependencies into beans effortlessly, reducing the manual configuration overhead. This promotes a more modular and maintainable codebase."*

12. Explain the role of the `WebTestClient` in Spring WebFlux testing.

Testing is a crucial aspect of software development, and understanding how to use `WebTestClient` is important for testing reactive applications in Spring WebFlux.

**How to answer:** Explain that `WebTestClient` is a testing utility provided by Spring WebFlux for testing web applications. It allows for making HTTP requests and asserting the responses, making it essential for testing the behavior of reactive controllers.

**Example Answer:** *"The `WebTestClient` in Spring WebFlux is a testing utility that facilitates testing of web applications. It enables us to make HTTP requests to our reactive controllers and assert the responses, ensuring that the application behaves as expected. This is particularly useful for testing the reactive nature of our endpoints."*

13. How does Spring WebFlux handle content negotiation?

Content negotiation is crucial for supporting various data formats in a web application. Understanding how Spring WebFlux handles content negotiation is important for building versatile APIs.

**How to answer:** Explain that Spring WebFlux uses the `ContentNegotiationConfigurer` to handle content negotiation. The `produces` and `consumes` attributes in the `@RequestMapping` annotation play a role in specifying the supported media types.

**Example Answer:** *"Spring WebFlux handles content negotiation through the `ContentNegotiationConfigurer`, which allows us to configure how the application produces and consumes different media types. The `produces` and `consumes` attributes in the `@RequestMapping` annotation are used to specify the supported media types for a particular endpoint."*

14. What is the purpose of the `@ModelAttribute` annotation in Spring WebFlux?

The `@ModelAttribute` annotation plays a role in Spring WebFlux for binding method parameters to model attributes. Understanding its purpose is crucial for handling data flow between the client and server.

**How to answer:** Explain that `@ModelAttribute` is used to bind method parameters to model attributes, allowing data transfer between the client and server. It is often used in conjunction with form submissions or when passing data to views.

**Example Answer:** *"In Spring WebFlux, the `@ModelAttribute` annotation is used to bind method parameters to model attributes. This facilitates the transfer of data between the client and server, commonly employed when handling form submissions or passing data to views."*

15. How can you enable CORS (Cross-Origin Resource Sharing) in a Spring WebFlux application?

Enabling CORS is crucial for allowing cross-origin requests in web applications. Understanding how to enable CORS in Spring WebFlux is important for building web services that interact with multiple domains.

**How to answer:** Explain that CORS can be enabled in Spring WebFlux by using the `@CrossOrigin` annotation on controller methods or by configuring CORS globally using `CorsRegistry` in the `WebFluxConfigurer` interface.

**Example Answer:** *"To enable CORS in a Spring WebFlux application, we can use the `@CrossOrigin` annotation on specific controller methods. Alternatively, for global configuration, we can implement the `WebFluxConfigurer` interface and configure CORS using the `addMapping` method in the `CorsRegistry`."*

16. Explain the role of the `ReactiveCrudRepository` interface in Spring WebFlux.

The `ReactiveCrudRepository` interface is integral for performing CRUD (Create, Read, Update, Delete) operations in a reactive way. Understanding its role is essential for efficient data manipulation in Spring WebFlux applications.

**How to answer:** Explain that `ReactiveCrudRepository` is an interface provided by Spring Data that extends the `ReactiveSortingRepository`. It offers methods for performing standard CRUD operations on entities, making it easier to interact with a reactive database.

**Example Answer:** *"The `ReactiveCrudRepository` interface in Spring WebFlux is a part of Spring Data and extends `ReactiveSortingRepository`. It provides methods for performing CRUD operations on entities in a reactive manner. This interface simplifies the process of interacting with a reactive database, offering convenience methods for data manipulation."*

17. How does Spring WebFlux handle data validation?

Data validation is crucial for ensuring the integrity of input data in an application. Understanding how Spring WebFlux handles data validation is essential for building robust and secure applications.

**How to answer:** Explain that Spring WebFlux leverages the `@Validated` and `@Valid` annotations for validating data. Additionally, the `ValidationUtils` class can be used for programmatic validation within a handler method.

**Example Answer:** *"Spring WebFlux handles data validation through the use of annotations such as `@Validated` and `@Valid`. These annotations can be applied to method parameters or request bodies to ensure that the incoming data meets specified criteria. Additionally, the `ValidationUtils` class provides programmatic validation options within a handler method."*

18. Explain the role of the `WebFilter` interface in Spring WebFlux.

The `WebFilter` interface is essential for implementing cross-cutting concerns in a Spring WebFlux application. Understanding its role is crucial for applying filters to incoming requests and responses.

**How to answer:** Explain that the `WebFilter` interface allows the implementation of filters that can be applied to incoming requests and outgoing responses. Filters can perform tasks such as logging, authentication, or modifying the request and response.

**Example Answer:** *"In Spring WebFlux, the `WebFilter` interface enables the implementation of filters that can be applied to incoming requests and outgoing responses. Filters are powerful for handling cross-cutting concerns, such as logging, authentication, or making modifications to the request and response before they reach the handler."*

19. How can you handle file uploads in a Spring WebFlux application?

Handling file uploads is a common requirement in web applications. Understanding how to handle file uploads in a Spring WebFlux application is essential for building features that involve file processing.

**How to answer:** Explain that file uploads in Spring WebFlux can be handled using the `@RequestPart` annotation or by utilizing the `multipart/form-data` content type. Discuss the use of `FormData` and `FilePart` for processing file uploads.

**Example Answer:** *"To handle file uploads in a Spring WebFlux application, we can use the `@RequestPart` annotation or process the `multipart/form-data` content type. The `FormData` class helps in extracting form data, and `FilePart` is specifically designed for handling file uploads, making it straightforward to process and save uploaded files."*

20. Explain the concept of reactive programming in the context of Spring WebFlux.

Reactive programming is at the core of Spring WebFlux, and understanding its principles is essential for building responsive and scalable applications.

**How to answer:** Explain that reactive programming is an asynchronous programming paradigm that focuses on handling and processing streams of data. In Spring WebFlux, this involves the use of reactive types like `Flux` and `Mono` to represent and process asynchronous data streams in a non-blocking manner.

**Example Answer:** *"Reactive programming in Spring WebFlux is centered around the asynchronous handling of data streams. It utilizes reactive types such as `Flux` for handling multiple values and `Mono` for at most one value. This approach allows for non-blocking and responsive handling of data, making it well-suited for scenarios with high concurrency."*

21. How does Spring WebFlux support server-sent events (SSE)?

Server-sent events (SSE) enable server-to-client communication over a single HTTP connection. Understanding how Spring WebFlux supports SSE is crucial for building real-time features in web applications.

**How to answer:** Explain that Spring WebFlux supports SSE through the `MediaType.TEXT\_EVENT\_STREAM\_VALUE` media type and the `Sinks.Many` class for emitting server-sent events. Discuss how SSE can be implemented in controller methods using the `Flux` return type.

**Example Answer:** *"Spring WebFlux supports server-sent events (SSE) using the `MediaType.TEXT\_EVENT\_STREAM\_VALUE` media type. The `Sinks.Many` class is used for emitting SSE, and in controller methods, we can return a `Flux` of events to be streamed to clients in real-time."*

22. What is the purpose of the `@ExceptionHandler` annotation in Spring WebFlux?

Error handling is a critical aspect of any application, and the `@ExceptionHandler` annotation plays a key role in handling exceptions in Spring WebFlux.

**How to answer:** Explain that `@ExceptionHandler` is used to handle exceptions at the global level in a Spring WebFlux application. It allows developers to define methods that will be invoked when specific exceptions occur, providing a centralized way to manage errors.

**Example Answer:** *"The `@ExceptionHandler` annotation in Spring WebFlux is used to handle exceptions at the global level. By annotating methods with this annotation, developers can define how the application should respond when specific exceptions occur. This provides a centralized and organized approach to error handling."*

23. How can you implement caching in a Spring WebFlux application?

Caching is a valuable technique for improving application performance, and understanding how to implement caching in Spring WebFlux is important for optimizing data retrieval and processing.

**How to answer:** Explain that caching in Spring WebFlux can be implemented using the `@Cacheable` and related annotations. Discuss the use of caching configuration and the `CacheManager` interface for defining caching behavior.

**Example Answer:** *"In Spring WebFlux, caching can be implemented using annotations like `@Cacheable`. By applying these annotations to methods, we can specify caching behavior. Additionally, configuring caching settings and using the `CacheManager` interface allows us to control how caching is managed in the application."*

24. How does Spring WebFlux handle transaction management?

Transaction management is essential for ensuring data integrity in database operations. Understanding how Spring WebFlux handles transactions is crucial for building reliable and consistent applications.

**How to answer:** Explain that Spring WebFlux uses the `@Transactional` annotation for managing transactions. Transactions can be configured at the method level, and Spring WebFlux leverages reactive transactional operators for handling transactions in a reactive environment.

**Example Answer:** *"Spring WebFlux handles transaction management using the `@Transactional` annotation. By annotating methods with `@Transactional`, we can define the scope and behavior of transactions. In a reactive environment, Spring WebFlux leverages reactive transactional operators to ensure proper transaction handling."*