8

Features of the framework for building non-blocking RESTful Service

Java Backend Developer I

Objetivos

Comprender los conceptos:

- Spring Web Flux
  - DispatcherHandler
  - Annotated Controllers
  - Functional Endpoints
  - Reactive Core
  - URI Links
  - HTTP Caching
  - HTTP/2
- WebSockets
- RSocket
- Reactive Libraries

Agenda

Revisión de los siguientes conceptos:

- Spring Web Flux
  - DispatcherHandler
  - Annotated Controllers
  - Functional Endpoints
  - Reactive Core
  - URI Links
  - HTTP Caching
  - HTTP/2
- WebSockets
- RSocket
- Reactive Libraries

Spring Web Flux

Concepto

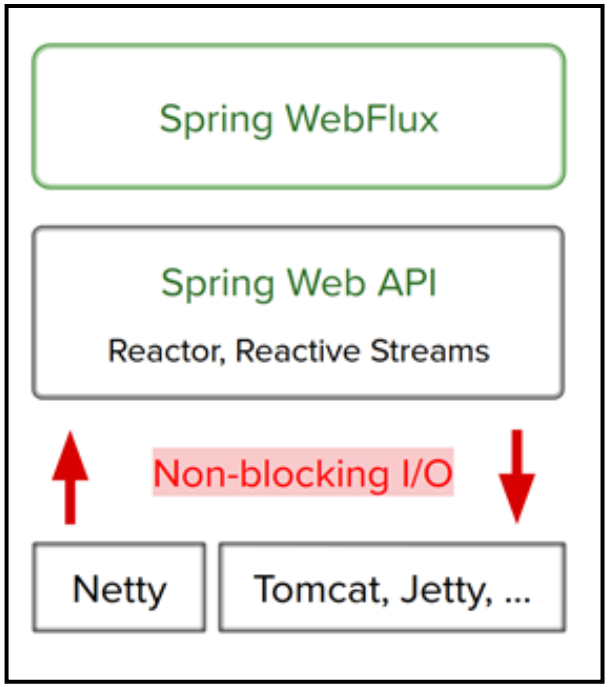
Spring WebFlux, de manera similar a Spring MVC, está diseñado alrededor del patrón del front controller frontal, donde un WebHandler central, conocido como DispatcherHandler, proporciona un algoritmo compartido para el procesamiento de solicitudes, mientras que el trabajo real se realiza mediante componentes delegables configurables. Este modelo es flexible y admite diversos workflows.

Spring Framework 5 incluye un nuevo módulo spring-webflux. El módulo contiene soporte para clientes HTTP y WebSocket reactivos, así como para aplicaciones web de servidores reactivos, incluyendo REST, navegador HTML e interacciones de estilo WebSocket.

Spring Web Flux

Concepto

Spring WebFlux, se agregó en la versión 5.0. el cual admite la backpressure de Reactive Streams y se ejecuta en servidores como contenedores Netty, Undertow y Servlet 3.1+.



Spring Web Flux

DispatcherHandler

Concepto

DispatcherHandler descubre los componentes delegados que necesita la configuración de Spring. También está diseñado para ser un bean Spring en sí mismo e implementa ApplicationContextAware para acceder al contexto en el que se ejecuta.

Si DispatcherHandler se declara con un nombre de bean webHandler, a su vez, WebHttpHandlerBuilder el cual reúne una cadena de procesamiento de solicitudes, como se describe en la API de WebHandler.

Spring Web Flux

DispatcherHandler

Concepto

La configuración de Spring en una aplicación WebFlux generalmente contiene:

- DispatcherHandler con el nombre de bean webHandler.
- WebFilter y WebExceptionHandler beans.
- DispatcherHandler beans especiales.
- Otros necesarios.

Spring Web Flux

Annotated Controllers

Concepto

Spring WebFlux proporciona un modelo de programación basado en anotaciones, donde los componentes @Controller y @RestController usan anotaciones para expresar asignaciones de solicitudes, entradas de solicitudes, manejar excepciones y más. Los controladores anotados no tienen que extender las clases base ni implementar interfaces específicas.

@RestController

Es una anotación compuesta que se conjuga con @Controller y @ResponseBody para indicar a un controlador cuyo método hereda la anotación @ResponseBody de nivel de tipo y, por lo tanto, escribe directamente en el cuerpo de la respuesta frente a la resolución de la vista y la representación con un Plantilla HTML.

Spring Web Flux

Annotated Controllers

@RequestMapping

Puede usar la anotación @RequestMapping para asignar solicitudes a métodos de controladores. Tiene varios atributos para hacer coincidir por URL, método HTTP, parámetros de solicitud, encabezados y tipos de medios.

Hay tambien metodos HTTP especificos variantes of @RequestMapping

@GetMapping

@PostMapping

@PutMapping

@DeleteMapping

@PatchMapping

Spring Web Flux

Functional Endpoints

Concepto

Spring WebFlux incluye WebFlux.fn, un modelo de programación funcional liviano en el que las funciones se utilizan para enrutar y manejar solicitudes y los contracts están diseñados para la inmutabilidad. Es una alternativa al modelo de programación basado en anotaciones, pero de lo contrario se ejecuta en la misma base Reactive Core.

En WebFlux.fn, una solicitud HTTP se maneja con una **HandlerFunction** una función que toma ServerRequest y devuelve un ServerResponse retrasado (es decir, Mono <ServerResponse>).

Spring Web Flux

Functional Endpoints

Tanto la solicitud como el objeto de respuesta tienen contratos inmutables que ofrecen acceso compatible con JDK 8 a la solicitud y respuesta HTTP. HandlerFunction es el equivalente del cuerpo de un método @RequestMapping en el modelo de programación basado en anotaciones.

Las solicitudes entrantes se enrutan a una función de controlador con una Función de enrutador: una función que toma ServerRequest y devuelve una Función de controlador retrasada (es decir, Mono <Función de controlador>). Cuando la función del enrutador coincide, se devuelve una función de controlador; de lo contrario, un Mono vacío.

Spring Web Flux

Functional Endpoints

RouterFunction es el equivalente de una anotación @RequestMapping, pero con la gran diferencia de que las funciones del enrutador proporcionan no solo datos, sino también comportamiento.

RouterFunctions.route () proporciona un generador de enrutadores que facilita la creación de enrutadores, como muestra el siguiente ejemplo:

Spring Web Flux

Reactive core

El módulo spring-web contiene el siguiente soporte básico para aplicaciones web reactivas:

Para el procesamiento de solicitudes del servidor hay dos niveles de soporte.

HttpHandler Contrato básico para el manejo de solicitudes HTTP con E / S sin bloqueo y contrapresión de corrientes reactivas, junto con adaptadores para Reactor Netty, Undertow, Tomcat, Jetty y cualquier contenedor Servlet 3.1+.

WebHandler API API web de uso general de nivel ligeramente superior para el manejo de solicitudes, además de la cual se construyen modelos de programación concretos, como controladores anotados y puntos finales funcionales.

Spring Web Flux

Reactive core

Para el lado del cliente, existe un contrato básico **ClientHttpConnector** para realizar solicitudes HTTP con E / S sin bloqueo y contrapresión de Corrientes reactivas, junto con adaptadores para Reactor Netty y para el Reactivo Jetty HttpClient. El WebClient de nivel superior utilizado en las aplicaciones se basa en este contrato básico.

Para el cliente y el servidor, códecs para la serialización y deserialización del contenido de respuesta y solicitud HTTP.

Spring Web Flux

URI Links

UriComponents

UriComponentsBuilder ayuda a construir URI a partir de templates de URI con variables, como muestra el siguiente ejemplo en Java:

```
UriComponents uriComponents = UriComponentsBuilder
        .fromUriString("https://example.com/hotels/{hotel}")  1
        .queryParam("q", "{q}")  2
        .encode()  3
        .build();  4

URI uri = uriComponents.expand("Westin", "123").toUri();  5
```

Spring Web Flux

URI Links

UriBuilder

UriComponentsBuilder implementa UriBuilder.

Puede crear un UriBuilder, a su vez, con un UriBuilderFactory. Juntos, UriBuilderFactory y UriBuilder proporcionan un mecanismo conectable para crear URI a partir de plantillas de URI, basadas en una configuración compartida, como una URL base, preferencias de codificación y otros detalles.

Puede configurar RestTemplate y WebClient con UriBuilderFactory para personalizar la preparación de URI. DefaultUriBuilderFactory es una implementación predeterminada de UriBuilderFactory que usa UriComponentsBuilder internamente y expone las opciones de configuración compartidas.

Spring Web Flux

URI Links

UriBuilder

Ejemplo para configurar RestTemplate

```
// import org.springframework.web.util.DefaultUriBuilderFactory.EncodingMode;

String baseUrl = "https://example.org";
DefaultUriBuilderFactory factory = new DefaultUriBuilderFactory(baseUrl);
factory.setEncodingMode(EncodingMode.TEMPLATE_AND_VALUES);

RestTemplate restTemplate = new RestTemplate();
restTemplate.setUriTemplateHandler(factory);
```

Spring Web Flux

URI Links

URI Encoding

UriComponentsBuilder expone las opciones de codificación en dos niveles:

UriComponentsBuilder # encode (): codifica previamente la plantilla URI primero y luego codifica estrictamente las variables URI cuando se expande.

UriComponents # encode (): codifica los componentes de URI después de expandir las variables de URI.

Ambas opciones reemplazan caracteres no ASCII e ilegales con octetos escapados. Sin embargo, la primera opción también reemplaza los caracteres con significado reservado que aparecen en las variables URI.

Spring Web Flux

URI Links

URI Encoding

Para la mayoría de los casos, es probable que la primera opción dé el resultado esperado, ya que trata las variables URI como datos para codificarlas completamente, mientras que la opción 2 es útil solo si las variables URI contienen intencionalmente caracteres reservados.

El siguien

```
URI uri = UriComponentsBuilder.fromPath("/hotel list/{city}")
        .queryParam("q", "{q}")
        .encode()
        .buildAndExpand("New York", "foo+bar")
        .toUri();

// Result is "/hotel%20list/New%20York?q=foo%2Bbar"
```

Spring Web Flux

HTTP Caching

Concepto

El HTTP catching puede mejorar significativamente el rendimiento de una aplicación web. El HTTP catching gira en torno al encabezado de respuesta de Control de caché y los encabezados de solicitud condicional posteriores, como Last-Modified y ETag.

Cache-Control aconseja a los cachés privados (por ejemplo, navegador) y públicos (por ejemplo, proxy) cómo almacenar en caché y reutilizar las respuestas.

Un encabezado ETag se usa para hacer una solicitud condicional que puede resultar en un 304 (NO MODIFICADO) sin cuerpo, si el contenido no ha cambiado. ETag puede verse como un sucesor más sofisticado del encabezado Last-Modified.

Spring Web Flux

HTTP/2

Concepto

HTTP / 2 es compatible con Reactor Netty, Tomcat, Jetty y Undertow. Sin embargo, hay consideraciones relacionadas con la configuración del servidor. Para más detalles, vea la página wiki HTTP / 2.

Consideraciones necesarias

La especificación HTTP / 2 y las implementaciones del navegador traen nuevas restricciones de seguridad en comparación con las aplicaciones HTTP / 1.1 seguras existentes:

Spring Web Flux

HTTP/2

Consideraciones necesarias

- TLS 1.2, SNI y ALPN, requisitos para actualizar al protocolo HTT / 2
- Un certificado de servidor seguro, generalmente con un algoritmo de firma fuerte y una clave de 2048+ bits
- Todos los requisitos de TLS enumerados en la especificación HTTP / 2

TLS 1.2 no es compatible de forma nativa con JDK8, pero está en JDK9; Además, las implementaciones alternativas de TLS (incluidos los enlaces nativos) son populares porque pueden ofrecer ganancias de rendimiento en comparación con la pila JDK.

WebSockets

Concepto

El protocolo WebSocket, RFC 6455, proporciona una forma estandarizada de establecer un canal de comunicación bidireccional de dúplex completo entre el cliente y el servidor a través de una única conexión TCP. Es un protocolo TCP diferente de HTTP, pero está diseñado para funcionar sobre HTTP, utilizando los puertos 80 y 443 y permitiendo la reutilización de las reglas de firewall existentes.

Una interacción de WebSocket comienza con una solicitud HTTP que utiliza el encabezado de actualización HTTP para actualizar o, en este caso, para cambiar al protocolo WebSocket. El siguiente ejemplo muestra tal interacción:

WebSockets

Ejemplo

```
GET /spring-websocket-portfolio/portfolio HTTP/1.1
Host: localhost:8080
Upgrade: websocket 1
Connection: Upgrade 2
Sec-WebSocket-Key: Uc9l9TMkWGbHFD2qnFHltg==
Sec-WebSocket-Protocol: v10.stomp, v11.stomp
Sec-WebSocket-Version: 13
Origin: http://localhost:8080
```

1. The `Upgrade` header.
2. Using the `Upgrade` connection.

WebSockets

HTTP Versus WebSocket

Aunque WebSocket está diseñado para ser compatible con HTTP y comienza con una solicitud HTTP, es importante comprender que los dos protocolos conducen a arquitecturas y modelos de programación de aplicaciones muy diferentes.

En HTTP y REST, una aplicación se modela con tantas URL. Para interactuar con la aplicación, los clientes acceden a esas URL, estilo de solicitud-respuesta. Los servidores enrutan las solicitudes al controlador apropiado en función de la URL, el método y los encabezados de HTTP.

Por el contrario, en WebSockets, generalmente solo hay una URL para la conexión inicial. Posteriormente, todos los mensajes de la aplicación fluyen en esa misma conexión TCP. Esto apunta a una arquitectura de mensajería asincrónica, controlada por eventos y completamente diferente.

WebSockets

HTTP Versus WebSocket

WebSocket también es un protocolo de transporte de bajo nivel que, a diferencia de HTTP, no prescribe ninguna semántica al contenido de los mensajes. Eso significa que no hay forma de enrutar o procesar un mensaje a menos que el cliente y el servidor acuerden la semántica del mensaje.

Los clientes y servidores de WebSocket pueden negociar el uso de un protocolo de mensajería de nivel superior (por ejemplo, STOMP), a través del encabezado Sec-WebSocket-Protocol en la solicitud de protocolo de enlace HTTP. En ausencia de eso, necesitan crear sus propias convenciones.

WebSockets

Cuando usar WebSockets

WebSockets puede hacer que una página web sea dinámica e interactiva. Sin embargo, en muchos casos, una combinación de transmisión Ajax y HTTP o sondeo largo puede proporcionar una solución simple y efectiva.

Por ejemplo, las noticias, el correo y las redes sociales deben actualizarse dinámicamente, pero puede estar perfectamente bien hacerlo cada pocos minutos. La colaboración, los juegos y las aplicaciones financieras, por otro lado, deben estar mucho más cerca del tiempo real.

La latencia por sí sola no es un factor decisivo. Si el volumen de mensajes es relativamente bajo (por ejemplo, monitoreando fallas de la red) la transmisión o sondeo HTTP puede proporcionar una solución efectiva. Es la combinación de baja latencia, alta frecuencia y alto volumen lo que constituye el mejor caso para el uso de WebSocket.

WebSockets

Cuando usar WebSockets

Tenga en cuenta también que a través de Internet, los servidores proxy restrictivos que están fuera de su control pueden impedir las interacciones de WebSocket, ya sea porque no están configurados para pasar el encabezado de actualización o porque cierran conexiones de larga duración que parecen inactivas. Esto significa que el uso de WebSocket para aplicaciones internas dentro del firewall es una decisión más directa que para las aplicaciones públicas.

RSocket

Concepto

RSocket es un protocolo de aplicación para comunicación multiplexada y dúplex a través de TCP, WebSocket y otros transportes de bytes, utilizando uno de los siguientes modelos de interacción:

- Request-Response: envíe un mensaje y reciba uno de vuelta.
- Request-Stream: envíe un mensaje y reciba una secuencia de mensajes.
- Channel: envía secuencias de mensajes en ambas direcciones.
- Fire-and-Forget: envía un mensaje unidireccional.

RSocket

Concepto

Una vez que se realiza la conexión inicial, se pierde la distinción "cliente" frente a "servidor", ya que ambos lados se vuelven simétricos y cada lado puede iniciar una de las interacciones anteriores. Es por eso que en el protocolo se llama a los participantes "requester" y "responder", mientras que las interacciones anteriores se denominan "secuencias de solicitud" o simplemente "solicitudes".

RSocket

Implementacion en JAVA

La implementación de Java para RSocket se basa en Project Reactor. Los transportes para TCP y WebSocket están construidos en Reactor Netty. Como una biblioteca de Reactive Streams, Reactor simplifica el trabajo de implementar el protocolo. Para aplicaciones, es natural usar Flux y Mono con operadores declarativos y soporte transparente de contrapresión.

La API en RSocket Java es intencionalmente mínima y básica. Se centra en las características del protocolo y deja el modelo de programación de la aplicación (p. Ej., Codegen RPC frente a otros) como una preocupación independiente de nivel superior.

RSocket

Implementacion en JAVA

El contrato principal io.rsocket.RSocket modela los cuatro tipos de interacción de solicitud con Mono que representa una promesa para un solo mensaje, Flux un flujo de mensajes y io.rsocket. Cargue el mensaje real con acceso a datos y metadatos como buffers de bytes. El contrato RSocket se usa simétricamente. Para solicitar, la aplicación recibe un RSocket para realizar solicitudes. Para responder, la aplicación implementa RSocket para manejar solicitudes.

Reactive Libraries

Concepto

Spring WebFlux depende de **reactor-core** y lo usa internamente para componer una lógica asincrónica y proporcionar soporte de streams Reactivas.

En general, las API de WebFlux devuelven Flux o Mono (ya que se usan internamente) y aceptan indulgente cualquier implementación de Reactive Streams Publisher como entrada. El uso de Flux versus Mono es importante porque ayuda a expresar la cardinalidad, por ejemplo, si se esperan valores asíncronos únicos o múltiples, y eso puede ser esencial para tomar decisiones (por ejemplo, al codificar o decodificar mensajes HTTP).

Reactive Libraries

Concepto

Para controller anotados, WebFlux se adapta transparentemente a la biblioteca reactiva elegida por la aplicación.

Esto se hace con la ayuda de **ReactiveAdapterRegistry**, que proporciona soporte conectable para la biblioteca reactiva y otros tipos asíncronos. El registro tiene soporte incorporado para RxJava y CompletableFuture, pero también puede registrar otros.

Reactive Libraries

Concepto

Para las API funcionales (como los puntos finales funcionales, el cliente web y otros), se aplican las reglas generales para las API de WebFlux: Flux y Mono como valores de retorno y un editor de secuencias reactivas como entrada. Cuando se proporciona un publicador, ya sea personalizado o de otra biblioteca reactiva, solo se puede tratar como una secuencia con una semántica desconocida (0..N). Sin embargo, si se conoce la semántica, puede envolverla con Flux o Mono.from (Publisher) en lugar de pasar el Publisher sin formato.

Reactive Libraries

Concepto

Por ejemplo, dado un editor que no es mono, el escritor de mensajes Jackson JSON espera múltiples valores. Si el tipo de medio implica una secuencia infinita (por ejemplo, aplicación / json + secuencia), los valores se escriben y se vacían individualmente. De lo contrario, los valores se almacenan en una lista y se representan como una matriz JSON.

Lecturas adicionales

Para obtener información adicional, puede consultar los siguientes enlaces:

https://docs.spring.io/spring/docs/current/spring-framework-reference/web-reactive.html

Resumen

En este capítulo, usted aprendió:

- Spring Web Flux
  - DispatcherHandler
  - Annotated Controllers
  - Functional Endpoints
  - Reactive Core
  - URI Links
  - HTTP Caching
  - HTTP/2
- WebSockets
- RSocket
- Reactive Libraries