(/)

Configuración de Netty Reactor de arranque de primavera

Última modificación: 28 de marzo de 2019

> por baeldung (https://www.baeldung.com/author/baeldung/) (https://www.baeldung.com/author/baeldung/)

Bota de primavera (https://www.baeldung.com/category/spring/spring-boot/)

1. Información general

En este tutorial, veremos diferentes opciones de configuración para un servidor Reactor Netty en una aplicación Spring Boot. Al final, tendremos una aplicación que muestra diferentes enfoques de configuración.

2. ¿Qué es Reactor Netty?

Antes de comenzar, veamos qué es Reactor Netty y cómo se relaciona con Spring Boot.

Reactor Netty es un **marco de aplicación de red basado en eventos asíncronos** (https://projectreactor.io/docs/netty/snapshot/reference/index.html#getting-started) . Proporciona servidores y clientes TCP, HTTP y UDP listos para la contrapresión y sin bloqueo. Como su nombre lo indica, se basa en el marco de Netty (https://www.baeldung.com/netty) .

Ahora, veamos dónde entran la primavera y la primavera en la imagen.

Spring WebFlux (https://www.baeldung.com/spring-webflux) forma parte del marco de Spring y proporciona soporte de programación reactiva para aplicaciones web. Si estamos usando WebFlux en una aplicación **Spring Boot** , **Spring Boot configura automáticamente Reactor Netty como el servidor predeterminado** . Además de eso, podemos agregar explícitamente Reactor Netty a nuestro proyecto, y Spring Boot debería configurarlo de nuevo automáticamente.

Ahora, crearemos una aplicación para aprender cómo podemos personalizar nuestro servidor Reactor Netty de configuración automática. Después de eso, cubriremos algunos escenarios de configuración comunes.

3. Dependencias

En primer lugar, agregaremos la dependencia requerida de Maven.

Para usar el servidor Reactor Netty, agregaremos el *spring-boot-starter-webflux* (https://search.maven.org/search?q=g:org.springframework.boot%20a:spring-boot-starter-webflux) como una dependencia en nuestro archivo pom:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-webflux</artifactId>
    <version>2.1.3.RELEASE</version>
</dependency>
```

Esto también *atraerá el arranque de resortes-arranque-reactor-netty* (https://search.maven.org/search?q=g:org.springframework.boot%20a:spring-boot-starter-reactor-netty) como una dependencia transitiva en nuestro proyecto.

4. Configuración del servidor

4.1. Usando archivos de propiedades

Como primera opción, podemos configurar el servidor Netty a través de archivos de propiedades. Spring Boot expone algunas de las configuraciones comunes del servidor en el archivo de propiedades de la *aplicación* :

Vamos a definir el puerto del servidor en *application.properties* :

```
server.port=8088
```

O podríamos haber hecho lo mismo en *application.yml* :

```
server:
    port: 8088
```

Además del puerto del servidor, Spring Boot tiene muchas otras opciones de configuración del servidor (https://www.baeldung.com/spring-boot-application-configuration) disponibles . **Las propiedades que comienzan con el prefijo del *servidor* nos permiten anular la configuración predeterminada del servidor** . Podemos buscar fácilmente estas propiedades en la documentación de Spring (https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html) en la sección *CONFIGURACIÓN DEL SERVIDOR INCORPORADO .*

4.2. Uso de la configuración programática

Ahora, veamos cómo **podemos configurar nuestro servidor Netty integrado a través del código** . Para este propósito, Spring Boot nos da las clases *WebServerFactoryCustomizer* y *NettyServerCustomizer* .

Usemos estas clases para configurar el puerto Netty (https://www.baeldung.com/spring-boot-change-port) como hicimos anteriormente con nuestro archivo de propiedades (https://www.baeldung.com/properties-with-spring) :

```
@Component
public class NettyWebServerFactoryPortCustomizer
  implements WebServerFactoryCustomizer<NettyReactiveWebServerFactory>

    @Override
    public void customize(NettyReactiveWebServerFactory serverFactory)
        serverFactory.setPort(8088);
    }
}
```

Spring Boot recogerá nuestro componente de personalización de fábrica durante el inicio y configurará el puerto del servidor.

Alternativamente, podemos implementar *NettyServerCustomizer* :

```
private static class PortCustomizer implements NettyServerCustomizer
    private final int port;

    private PortCustomizer(int port) {
        this.port = port;
    }
    @Override
    public HttpServer apply(HttpServer httpServer) {
        return httpServer.port(port);
    }
}
```

Y añádelo a la fábrica del servidor:

```
serverFactory.addServerCustomizers(new ProtocolCustomizer(8088));
```

Estos dos enfoques nos dan mucha flexibilidad al configurar nuestro servidor Reactor Netty incorporado.

Además, también podemos acceder a la clase *ServerBootstrap* desde el marco de Netty y hacer nuestras personalizaciones allí:

```
private static class EventLoopNettyCustomizer implements NettyServer(

    @Override
    public HttpServer apply(HttpServer httpServer) {
        EventLoopGroup parentGroup = new NioEventLoopGroup();
        EventLoopGroup childGroup = new NioEventLoopGroup();
        return httpServer.tcpConfiguration(tcpServer -> tcpServer
          .bootstrap(serverBootstrap -> serverBootstrap
            .group(parentGroup, childGroup)
            .channel(NioServerSocketChannel.class)));
    }
}
```

Sin embargo, hay una advertencia para este caso. Como Spring Boot configura automáticamente el servidor Netty, **es posible que debamos omitir la configuración automática definiendo explícitamente nuestro bean *NettyReactiveWebServerFactory* .**

Para este propósito, deberíamos definir nuestro bean en una clase de configuración y agregar nuestro personalizador allí:

```
@Bean
public NettyReactiveWebServerFactory nettyReactiveWebServerFactory() {
    NettyReactiveWebServerFactory webServerFactory = new NettyReactive
    webServerFactory.addServerCustomizers(new EventLoopNettyCustomizer
    return webServerFactory;
}
```

A continuación, continuaremos con algunos escenarios comunes de configuración de Netty.

5. Configuración SSL

Veamos cómo podemos configurar SSL.

Usaremos la clase *SslServerCustomizer* que es otra implementación de *NettyServerCustomizer* :

```
@Component
public class NettyWebServerFactorySslCustomizer
  implements WebServerFactoryCustomizer<NettyReactiveWebServerFactory

    @Override
    public void customize(NettyReactiveWebServerFactory serverFactory
        Ssl ssl = new Ssl();
        ssl.setEnabled(true);
        ssl.setKeyStore("classpath:sample.jks");
        ssl.setKeyAlias("alias");
        ssl.setKeyPassword("password");
        ssl.setKeyStorePassword("secret");
        Http2 http2 = new Http2();
        http2.setEnabled(false);
        serverFactory.addServerCustomizers(new SslServerCustomizer(ss
        serverFactory.setPort(8443);
    }
}
```

Aquí hemos definido nuestras propiedades relacionadas con el almacén de claves, hemos deshabilitado HTTP / 2 y hemos establecido el puerto en 8443.

6. Configuración de registro de acceso

Ahora, veremos cómo podemos configurar el registro de acceso usando Logback (https://www.baeldung.com/logback) .

Spring Boot nos permite configurar el registro de acceso en el archivo de propiedades de la aplicación para Tomcat, Jetty y Undertow. Sin embargo, Netty todavía no tiene este soporte.

Para habilitar el registro de acceso de Netty, **deberíamos establecer *-Dreactor.netty.http.server.accessLogEnabled = true*** cuando ***ejecutemos*** nuestra aplicación:

```
mvn spring-boot:run -Dreactor.netty.http.server.accessLogEnabled=true
```

7. Conclusión

En este artículo, hemos cubierto cómo configurar el servidor Reactor Netty en una aplicación Spring Boot.

En primer lugar, usamos las capacidades de configuración general basadas en la propiedad Spring Boot. Y luego, exploramos cómo configurar Netty mediante programación de una manera detallada.

Finalmente, el código fuente de este artículo está disponible en Github (https://github.com/eugenp/tutorials/tree/master/spring-5-webflux) .

Deja una respuesta

Start the discussion...

Suscribir

LAS CATEGORÍAS

PRIMAVERA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SPRING/)
DESCANSO (HTTPS://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SEGURIDAD (HTTPS://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCIA (HTTPS://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTPS://WWW.BAELDUNG.COM/CATEGORY/JSON/JACKSON/)
CLIENTE HTTP (HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIE

TUTORIAL DE JAVA "BACK TO BASICS" (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
TUTORIAL HTTPCLIENT 4 (/HTTPCLIENT-GUIDE)
DESCANSO CON TUTORIAL DE PRIMAVERA (/REST-WITH-SPRING-SERIES)
TUTORIAL DE PERSISTENCIA DE PRIMAVERA (/PERSISTENCE-WITH-SPRING-SERIES)
SEGURIDAD CON SPRING (/SECURITY-SPRING)

ACERCA DE

ACERCA DE BAELDUNG (/ABOUT)
LOS CURSOS (HTTPS://COURSES.BAELDUNG.COM)
TRABAJO DE CONSULTORÍA (/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
EL ARCHIVO COMPLETO (/FULL_ARCHIVE)
ESCRIBIR PARA BAELDUNG (/CONTRIBUTION-GUIDELINES)
EDITORES (/EDITORS)
NUESTROS COMPAÑEROS (/PARTNERS)
PUBLICIDAD EN BAELDUNG (/ADVERTISE)

TÉRMINOS DE SERVICIO (/TERMS-OF-SERVICE)
POLÍTICA DE PRIVACIDAD (/PRIVACY-POLICY)
INFORMACIÓN DE LA COMPAÑÍA (/BAELDUNG-COMPANY-INFO)
CONTACTO (/CONTACT)