(/)

Registro de llamadas de Spring WebClient

Última modificación: 7 de septiembre de 2019

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

Reactivo (https://www.baeldung.com/category/reactive/)
Primavera (https://www.baeldung.com/category/spring/) +

1. Información general

En este tutorial, vamos a mostrar cómo personalizar de primavera *cliente Web* (https://www.baeldung.com/spring-5-webclient) - un cliente HTTP reactiva - registre las peticiones y respuestas.

2. *WebClient*

WebClient es una interfaz reactiva y sin bloqueo para solicitudes HTTP, basada en Spring WebFlux (https://www.baeldung.com/spring-webflux) . Tiene una API funcional y fluida con tipos reactivos para la composición declarativa.

Detrás de escena, *WebClient* llama a un cliente HTTP. Reactor Netty es el valor predeterminado y también se admite el *HttpClient* reactivo de Jetty. Además, es posible conectar otras implementaciones de cliente HTTP configurando un *ClientConnector* para *WebClient* .

3. Registro de solicitudes y respuestas

El valor por defecto *HttpClient* utilizado por *cliente Web* es la aplicación de Netty, por lo que **después cambiamos el *reactor.netty.http.client* nivel de registro en *DEBUG*,** podemos ver algunos solicitud de registro, pero si es necesario un registro personalizado, podemos CONFIGURACION *e* nuestros registradores a través *WebClient # filtros* (https://www.baeldung.com/spring-webclient-filters) :

```
WebClient
  .builder()
  .filters(exchangeFilterFunctions -> {
      exchangeFilterFunctions.add(logRequest());
      exchangeFilterFunctions.add(logResponse());
  })
  .build()
```

En este fragmento de código, hemos agregado dos filtros separados para registrar la solicitud y la respuesta.

Implementemos *logRequest* usando *ExchangeFilterFunction # ofRequestProcessor* :

```
ExchangeFilterFunction logRequest() {
    return ExchangeFilterFunction.ofRequestProcessor(clientRequest -> {
        if (log.isDebugEnabled()) {
            StringBuilder sb = new StringBuilder("Request: \n");
            //append clientRequest method and url
            clientRequest
              .headers()
              .forEach((name, values) -> values.forEach(value -> /* append header key/value */));
            log.debug(sb.toString());
        }
        return Mono.just(clientRequest);
    });
}
```

Cojín R

Ahora podemos cambiar el nivel de registro de *reactor.netty.http.client* a *INFO* o *ERROR* para tener una salida más limpia.

4. Solicitud de registro y respuesta con el cuerpo

Los clientes HTTP tienen características para registrar los cuerpos de solicitudes y respuestas. Por lo tanto, **para lograr el objetivo, vamos a utilizar un cliente HTTP con registro habilitado con nuestro cliente *web*.**

Podemos hacer esto configurando manualmente *WebClient.Builder # clientConnector* - veamos con los clientes Jetty y Netty HTTP.

4.1. Iniciar sesión con Jetty *HttpClient*

Primero, agreguemos la dependencia Maven para *jetty-reactive-httpclient* (https://search.maven.org/search?q=a:jetty-reactive-httpclient) a nuestro pom:

```
<dependency>
    <groupId>org.eclipse.jetty</groupId>
    <artifactId>jetty-reactive-httpclient</artifactId>
    <version>1.0.3</version>
</dependency>
```

Luego, crearemos un Jetty *HttpClient* personalizado :

```
SslContextFactory.Client sslContextFactory = new SslContextFactory.Client();
HttpClient httpClient = new HttpClient(sslContextFactory) {
    @Override
    public Request newRequest(URI uri) {
        Request request = super.newRequest(uri);
        return enhance(request);
    }
};
```

Aquí, hemos anulado *HttpClient # newRequest* , luego *hemos* envuelto la *solicitud* en un potenciador de registro. A continuación, debemos registrar eventos con la solicitud para que podamos iniciar sesión a medida que cada parte de la solicitud esté disponible:

```
Request enhance(Request request) {
    StringBuilder group = new StringBuilder();
    request.onRequestBegin(theRequest -> {
        // append request url and method to group
    });
    request.onRequestHeaders(theRequest -> {
        for (HttpField header : theRequest.getHeaders()) {
            // append request headers to group
        }
    });
    request.onRequestContent((theRequest, content) -> {
        // append content to group
    });
    request.onRequestSuccess(theRequest -> {
        log.debug(group.toString());
        group.delete(0, group.length());
    });
    group.append("\n");
```



```
            // append response headers to group
        }
    });
    request.onResponseContent((theResponse, content) -> {
        // append content to group
    });
    request.onResponseSuccess(theResponse -> {
        log.debug(group.toString());
    });
    return request;
}
```

Finalmente, tenemos que construir la instancia de *WebClient* :

```
WebClient
  .builder()
  .clientConnector(new JettyClientHttpConnector(httpClient))
  .build()
```

Por supuesto, como lo hicimos antes, necesitaremos establecer el nivel de registro de *RequestLogEnhancer* en *DEBUG* .

4.2. Iniciar sesión con Netty *HttpClient*

Primero, creemos un Netty *HttpClient* :

```
HttpClient httpClient = HttpClient
  .create()
  .wiretap(true)
```

Una vez habilitada la intervención telefónica, cada solicitud y respuesta se registrarán con todo detalle.

A continuación, tenemos que establecer el nivel de registro del paquete de cliente de Netty *reactor.netty.http.client* en *DEBUG* :

```
logging.level.reactor.netty.http.client=DEBUG
```

Ahora, construyamos el *WebClient* :

```
WebClient
  .builder()
  .clientConnector(new ReactorClientHttpConnector(httpClient))
  .build()
```

Nuestro *WebClient* registrará cada solicitud y respuesta con todo detalle, **pero el formato predeterminado del registrador incorporado de Netty contiene tanto representación hexadecimal como texto de los cuerpos** y también una gran cantidad de datos sobre eventos de solicitud y respuesta.

Entonces, si necesitamos un registrador personalizado para Netty, podemos configurar el *HttpClient* :

〈 Cojín R

```
        b -> BootstrapHandlers.updateLogSupport(b, new CustomLogger(HttpClient.class)))
  .build()
```

Por último, implementemos nuestro *CustomLogger* que extiende *LoggingHandler* :

```
public class CustomLogger extends LoggingHandler {
    public CustomLogger(Class<?> clazz) {
        super(clazz);
    }

    @Override
    protected String format(ChannelHandlerContext ctx, String event, Object arg) {
        if (arg instanceof ByteBuf) {
            ByteBuf msg = (ByteBuf) arg;
            return decode(
              msg, msg.readerIndex(), msg.readableBytes(), defaultCharset());
        }
        return super.format(ctx, event, arg);
    }

    // further code omitted for brevity
}
```

5. Conclusión

En este tutorial, hemos utilizado varias técnicas para registrar datos de solicitudes y respuestas mientras usamos Spring *WebClient* .

Como siempre, el código está disponible en nuestro GitHub (https://github.com/eugenp/tutorials/tree/master/spring-5-reactive-client) .

Deja una respuesta

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/)
HTTP DEL LADO DEL CLIENTE (HTTPS://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTPS://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIE

TUTORIAL DE JAVA "VOLVER A LO BÁSICO" (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
HTTPCLIENT 4 TUTORIAL (/HTTPCLIENT-GUIDE)
RESTO CON SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
TUTORIAL SPRING PERSISTENCE (/PERSISTENCE-WITH-SPRING-SERIES)
SEGURIDAD CON PRIMAVERA (/SECURITY-SPRING)

ACERCA DE

SOBRE 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)
ANUNCIE EN BAELDUNG (/ADVERTISE)

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