EBook Gratis

APRENDIZAJE apache-camel

Free unaffiliated eBook created from
Stack Overflow contributors.

#apache-camel

Tabla de contenido

Acerca de ... 1

Capítulo 1: Empezando con el camello apache ... 2

Observaciones ... 2

Examples ... 2

Instalación o configuración ... 2

Dependencia de Maven ... 2

Gradle ... 3

Bota de primavera ... 3

Lenguaje específico de dominio de camello ... 3

Capítulo 2: Pruebas de integración en rutas existentes con Apache-Camel y Spring (y DBUnit ... 5

Introducción ... 5

Parámetros ... 5

Observaciones ... 5

Examples ... 6

Ejemplo de ruta en camello ... 6

Ejemplo de procesador de camello ... 6

Ejemplo de clase de prueba de integración de camellos ... 8

Capítulo 3: Pub / Sub usando Camel + Redis ... 12

Observaciones ... 12

Examples ... 12

RedisPublisher ... 12

RedisSubscriber ... 12

Contexto de primavera del suscriptor ... 13

Editor de contexto de primavera ... 13

ManagedCamel ... 14

Creditos ... 16

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: apache-camel

It is an unofficial and free apache-camel ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official apache-camel.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con el camello apache

Observaciones

Apache Camel es un marco que facilita principalmente la resolución de desafíos de integración empresarial. En su núcleo se puede considerar como un generador de motores de enrutamiento. En esencia, le permite conectar sistemas (puntos finales) a través de rutas. Estas rutas aceptan mensajes que pueden ser de cualquier tipo de datos.

El marco de Apache Camel también contiene un conjunto completo de EIP (patrones de integración empresarial) como divisor, agregadores, enrutamiento basado en contenido, etc. Dado que el marco se puede implementar en varias aplicaciones independientes en Java, en varios servidores de aplicaciones como WildFly y Tomcat o en un bus de servicios empresariales de pleno derecho, se puede ver como un marco de integración.

Para comenzar con el marco, debe agregarlo a un proyecto usando uno de los siguientes métodos:

1. Maven
2. Gradle
3. Bota de primavera
4. Referencia de biblioteca JAR antigua y simple agregada a su proyecto.

Examples

Instalación o configuración

Instrucciones detalladas para agregar las dependencias de Camel requeridas.

Dependencia de Maven

Una de las formas más comunes de incluir Apache Camel en su aplicación es a través de una dependencia de Maven. Al agregar el bloque de dependencia a continuación, Maven resolverá las bibliotecas y dependencias de Camel por usted.

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-core</artifactId>
  <version>2.17.3</version>
</dependency>
```

Gradle

Otra forma común de incluir Apache Camel en su aplicación es a través de una dependencia de Gradle. Simplemente agregue la línea de dependencia a continuación y Gradle importará la biblioteca Camel y sus dependencias por usted.

```
// https://mvnrepository.com/artifact/org.apache.camel/camel-core
compile group: 'org.apache.camel', name: 'camel-core', version: '2.17.3'
```

Bota de primavera

A partir de Camel 2.15, ahora puede aprovechar la dependencia Spring Boot de Apache Camel. La diferencia con esta biblioteca de Camel es que proporciona una configuración automática de opinión, incluida la detección automática de rutas de Camel.

```
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-spring-boot</artifactId>
    <version>${camel.version}</version> <!-- use the same version as your Camel core version -
->
</dependency>
```

Lenguaje específico de dominio de camello

El DSL (lenguaje específico del dominio) de Camel es una de las características que hace que Camel sobresalga de otros marcos de integración. Si bien otros marcos también incluyen un concepto de DSL, generalmente en forma de un archivo XML, en estos casos, el DSL siempre fue un lenguaje personalizado.

Camel ofrece múltiples DSL en lenguajes de programación como Java, Scala, Groovy y en XML.

Por ejemplo, una ruta de copia de archivos simple se puede hacer de varias maneras como se muestra en la lista a continuación

- Java DSL

  ```
  from("file:data/in").to("file:data/out");
  ```

- Blueprint / Spring DSL (XML)

  ```
  <route>
    <from uri="file:data/inbox"/>
    <to uri="file:data/out"/>
  </route>
  ```

- Scala DSL

```
from "file:data/inbox" -> "file:data/out"
```

Lea Empezando con el camello apache en línea: https://riptutorial.com/es/apache-camel/topic/3511/empezando-con-el-camello-apache

Capítulo 2: Pruebas de integración en rutas existentes con Apache-Camel y Spring (y DBUnit)

Introducción

El objetivo de este wiki es mostrarle cómo ejecutar pruebas de integración utilizando Apache Camel.

Más precisamente, al hacer esto, podrá iniciar una ruta existente de principio a fin (con o sin su base de datos real) o interceptar el intercambio entre cada parte de la ruta y probar si sus encabezados o cuerpo son correctos o no.

El proyecto en el que he estado haciendo esto usa Spring clásico con configuración xml y DBUnit para simular una base de datos de prueba. Espero que esto te dé algunas pistas.

Parámetros

Parámetro / Función	Detalles
Intercambiar	El intercambio se utiliza dentro del procesador de camellos para pasar objetos entre partes de su ruta
CamelContext	El contexto de camello se utiliza en la prueba para iniciar y detener manualmente el contexto.
ProducerTemplate	Le permite enviar mensajes en su ruta, configurando el intercambio completo manualmente o enviando encabezados / cuerpo ficticios
Consejos Con	Le ayuda a redefinir una ruta existente con el contexto actual
WeaveById	Utilizado dentro del consejo con configuración, le dice a las partes de su ruta cómo comportarse (también puede usar *weaveByToString*)
Punto de Mock	El mockendpoint es un punto que define para su prueba. En su weaveById, puede indicar su ruta a su procesamiento habitual y entrar en un punto de simulación en lugar de seguir la ruta habitual. De esta manera puede verificar el recuento de mensajes, el estado de intercambio ...

Observaciones

Algunas definiciones proporcionadas aquí no son perfectamente precisas, pero le ayudarán a entender el código anterior. Aquí hay algunos enlaces para obtener información más detallada:

- Sobre el uso de *AdviceWith* y *weaveById* (u otras formas de activar rutas), eche un vistazo a la documentación oficial de apache-camel: vea este enlace
- Sobre el uso de *ProducerTemplate* , vea nuevamente la documentación oficial: vea este enlace
- Para comprender realmente de qué se trata el camello: Documentación detallada de patrones de integración empresarial.

Esta forma particular de prueba es bastante difícil de encontrar, incluso en el desbordamiento de pila. Esto es bastante específico, pero no dude en pedir más detalles, tal vez pueda ayudar.

Examples

Ejemplo de ruta en camello

La siguiente ruta tiene un objetivo simple:

- Primero, verifica si el objeto **ImportDocumentProcess** está presente en la base de datos y lo agrega como un *encabezado de intercambio*
- Luego, agrega un **ImportDocumentTraitement** (que está vinculado al ImportDocumentProcess anterior) en la base de datos

Aquí está el código de esta ruta:

```
@Component
public class TestExampleRoute extends SpringRouteBuilder {

    public static final String ENDPOINT_EXAMPLE = "direct:testExampleEndpoint";

    @Override
    public void configure() throws Exception {
        from(ENDPOINT_EXAMPLE).routeId("testExample")
            .bean(TestExampleProcessor.class,
"getImportDocumentProcess").id("getImportDocumentProcess")
            .bean(TestExampleProcessor.class,
"createImportDocumentTraitement").id("createImportDocumentTraitement")
            .to("com.pack.camel.routeshowAll=true&multiline=true");
    }

}
```

El *ID* en las rutas no es obligatorio, también puede usar las cadenas de bean posteriormente. Sin embargo, creo que el uso de *ID* puede considerarse una buena práctica, en caso de que las cadenas de ruta cambien en el futuro.

Ejemplo de procesador de camello

El procesador solo contiene los métodos que necesita la ruta. Es solo un clásico Java Bean que contiene varios métodos. También puede *implementar Procesador* y anular el método de *proceso* .

Vea el código a continuación:

```
@Component("testExampleProcessor")
public class TestExampleProcessor {

    private static final Logger LOGGER = LogManager.getLogger(TestExampleProcessor.class);

    @Autowired
    public ImportDocumentTraitementServiceImpl importDocumentTraitementService;

    @Autowired
    public ImportDocumentProcessDAOImpl importDocumentProcessDAO;

    @Autowired
    public ImportDocumentTraitementDAOImpl importDocumentTraitementDAO;

    // ---- Constants to name camel headers and bodies
    public static final String HEADER_ENTREPRISE = "entreprise";

    public static final String HEADER_UTILISATEUR = "utilisateur";

    public static final String HEADER_IMPORTDOCPROCESS = "importDocumentProcess";

    public void getImportDocumentProcess(@Header(HEADER_ENTREPRISE) Entreprise entreprise,
Exchange exchange) {
        LOGGER.info("Entering TestExampleProcessor method : getImportDocumentProcess");

        Utilisateur utilisateur = SessionUtils.getUtilisateur();
        ImportDocumentProcess importDocumentProcess =
importDocumentProcessDAO.getImportDocumentProcessByEntreprise(
                entreprise);

        exchange.getIn().setHeader(HEADER_UTILISATEUR, utilisateur);
        exchange.getIn().setHeader(HEADER_IMPORTDOCPROCESS, importDocumentProcess);
    }

    public void createImportDocumentTraitement(@Header(HEADER_ENTREPRISE) Entreprise
entreprise,
            @Header(HEADER_UTILISATEUR) Utilisateur utilisateur,
            @Header(HEADER_IMPORTDOCPROCESS) ImportDocumentProcess importDocumentProcess,
Exchange exchange) {
        LOGGER.info("Entering TestExampleProcessor method : createImportDocumentTraitement");

        long nbImportTraitementBefore =
this.importDocumentTraitementDAO.countNumberOfImportDocumentTraitement();
        ImportDocumentTraitement importDocumentTraitement =
this.importDocumentTraitementService.createImportDocumentTraitement(
                entreprise, utilisateur, importDocumentProcess, "md5_fichier_example_test",
"fichier_example_test.xml");
        long nbImportTraitementAfter =
this.importDocumentTraitementDAO.countNumberOfImportDocumentTraitement();

        exchange.getIn().setHeader("nbImportTraitementBefore",
Long.valueOf(nbImportTraitementBefore));
        exchange.getIn().setHeader("nbImportTraitementAfter",
Long.valueOf(nbImportTraitementAfter));
```

```
        exchange.getIn().setHeader("importDocumentTraitement", importDocumentTraitement);
    }
// Rest of the code contains getters and setters for imported dependencies
}
```

No hay mucho que decir aquí, excepto que usamos el intercambio para transferir objetos de una parte a otra. Esta es la forma en que normalmente se hace en mi proyecto, ya que tenemos procesos muy complejos que manejar.

Ejemplo de clase de prueba de integración de camellos

> No olvide agregar el soporte de prueba de camello y el soporte de prueba de camello de primavera a las dependencias de su proyecto. Vea lo siguiente para usuarios expertos:

```
<dependency>
  <groupId>org.apache.camel</groupId>
  <artifactId>camel-test</artifactId>
  <version>${camel.version}</version>
  <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.apache.camel</groupId>
    <artifactId>camel-test-spring</artifactId>
    <version>${camel.version}</version>
     <scope>test</scope>
</dependency>
```

Esta clase activará y ejecutará pruebas en la ruta de ejemplo. Estas pruebas también usan **DBUnit** para simular una base de datos, aunque puede configurar su contexto para usar una base de datos real u otra clase de base simulada.

Primero, usamos una clase abstracta para compartir anotaciones comunes entre cada clase de prueba de integración de Camel que luego usaremos:

```
@RunWith(CamelSpringRunner.class)
@BootstrapWith(CamelTestContextBootstrapper.class)
@ContextConfiguration(locations = { "classpath:/test-beans.xml" })
@DbUnitConfiguration(dataSetLoader = ReplacementDataSetLoader.class)
@TestExecutionListeners({ DependencyInjectionTestExecutionListener.class,
DirtiesContextTestExecutionListener.class,
        DbUnitTestExecutionListener.class })
@DirtiesContext(classMode = ClassMode.AFTER_EACH_TEST_METHOD)
public abstract class AbstractCamelTI {

}
```

Tenga cuidado de no olvidar ninguna anotación o sus DAO no se inyectarán correctamente. Dicho esto, puede eliminar de forma segura las anotaciones DBUnit si no desea utilizar la base de datos que se muestra en su configuración de contexto.

EDITAR IMPORTANTE : He agregado el `@DirtiesContext(classMode =`

`ClassMode.AFTER_EACH_TEST_METHOD)` recientemente. De esa manera, el *contexto* del *camello* se vuelve a cargar para cada prueba. Realmente puedes probar cada parte de tu ruta individualmente. Sin embargo, si realmente desea eso, debe usar *remove ()* en las partes de la ruta elegida que no desea seguir. Algunos dirían que esto no es una prueba de integración real, y tendrían razón. Pero si, como yo, tiene procesadores grandes que necesita refactorizar, puede comenzar allí.

El siguiente código muestra el comienzo de la clase de prueba (ver abajo a continuación para las pruebas reales):

```
@DatabaseSetup(value = { "/db_data/dao/common.xml",
"/db_data/dao/importDocumentDAOCommonTest.xml" })
public class TestExampleProcessorTest extends AbstractCamelTI {

    @Autowired
    protected CamelContext camelContext;

    @EndpointInject(uri = "mock:catchTestEndpoint")
    protected MockEndpoint mockEndpoint;

    @Produce(uri = TestExampleRoute.ENDPOINT_EXAMPLE)
    protected ProducerTemplate template;

    @Autowired
    ImportDocumentTraitementDAO importDocumentTraitementDAO;

    // -- Variables for tests
    ImportDocumentProcess importDocumentProcess;

    @Override
    @Before
    public void setUp() throws Exception {
        super.setUp();

        importDocumentProcess = new ImportDocumentProcess();
        //specific implementation of your choice
    }
}
```

Se supone que la siguiente prueba debe activar la primera parte de la ruta y llevarla a un punto de `mockEndpoint` para que podamos probar si *ImportDocumentProcess* se ha seleccionado correctamente y se ha colocado en los encabezados:

```
@Test
public void processCorrectlyObtained_getImportDocumentProcess() throws Exception {
    camelContext.getRouteDefinitions().get(0).adviceWith(camelContext, new
AdviceWithRouteBuilder() {

        @Override
        public void configure() throws Exception {
            weaveById("getImportDocumentProcess").after().to(mockEndpoint);
        }
    });

    // -- Launching the route
    camelContext.start();
```

```
    template.sendBodyAndHeader(null, "entreprise", company);

    mockEndpoint.expectedMessageCount(1);
    mockEndpoint.expectedHeaderReceived(TestExampleProcessor.HEADER_UTILISATEUR, null);
    mockEndpoint.expectedHeaderReceived(TestExampleProcessor.HEADER_IMPORTDOCPROCESS,
importDocumentProcess);
    mockEndpoint.assertIsSatisfied();

    camelContext.stop();
}
```

La última prueba desencadena toda la ruta:

```
@Test
public void traitementCorrectlyCreated_createImportDocumentTraitement() throws Exception {
    camelContext.getRouteDefinitions().get(0).adviceWith(camelContext, new
AdviceWithRouteBuilder() {

        @Override
        public void configure() throws Exception {
            weaveById("createImportDocumentTraitement").after().to(mockEndpoint);
        }
    });

    // -- Launching the route
    camelContext.start();

    Exchange exchange = new DefaultExchange(camelContext);
    exchange.getIn().setHeader(TestExampleProcessor.HEADER_ENTREPRISE, company);
    exchange.getIn().setHeader(TestExampleProcessor.HEADER_UTILISATEUR, null); // No user in
this case
    exchange.getIn().setHeader(TestExampleProcessor.HEADER_IMPORTDOCPROCESS,
importDocumentProcess);

    long numberOfTraitementBefore =
this.importDocumentTraitementDAO.countNumberOfImportDocumentTraitement();

    template.send(exchange);

    mockEndpoint.expectedMessageCount(1);
    mockEndpoint.assertIsSatisfied();

    camelContext.stop();

    long numberOfTraitementAfter =
this.importDocumentTraitementDAO.countNumberOfImportDocumentTraitement();
    assertEquals(numberOfTraitementBefore + 1L, numberOfTraitementAfter);
}
```

También es posible redirigir la ruta actual a otro proceso. Pero prefiero redirigir a un punto de `mockEndpoint` . Es un poco más interesante porque realmente puedes hacer pruebas intermedias en tu cuerpo de intercambio y encabezados.

NOTA IMPORTANTE : En este ejemplo, estoy usando el siguiente código para obtener mis rutas y usar `adviceWith` con ellos:

```
camelContext.getRouteDefinitions().get(0).adviceWith(camelContext, new
AdviceWithRouteBuilder() { [...] });
```

SIN EMBARGO Es posible obtener la ruta mediante una ID previamente definida como una cadena, como esta:

```
 camelContext.getRouteDefinition("routeId").adviceWith(camelContext, new
AdviceWithRouteBuilder() {  [...]  });
```

Recomiendo encarecidamente este método, puede ahorrar mucho tiempo averiguar por qué fallan sus pruebas

Lea Pruebas de integración en rutas existentes con Apache-Camel y Spring (y DBUnit) en línea: https://riptutorial.com/es/apache-camel/topic/10630/pruebas-de-integracion-en-rutas-existentes-con-apache-camel-y-spring--y-dbunit-

Capítulo 3: Pub / Sub usando Camel + Redis

Observaciones

Usando el editor:

```
producerTemplate.asyncSendBody("direct:myprocedure", massageBody);
```

Usando el "createProducer ()" en ManagedCamel para crear el producerTemplate.

Examples

RedisPublisher

```
public class RedisPublisher extends RouteBuilder {

    public static final String CAMEL_REDIS_CHANNEL = "CamelRedis.Channel";
    public static final String CAMEL_REDIS_MESSAGE = "CamelRedis.Message";

    @Value("${redis.host}")
    private String redisHost;
    @Value("${redis.port}")
    private int redisPort;
    @Value("${redis.channel.mychannel}")
    private String redisChannel;

    private String producerName;

    @Required
    public void setProducerName(String producerName) {
        this.producerName = producerName;
    }

    @Override
    public void configure() throws Exception {
        from(producerName)
                .log(String.format("Publishing with redis in channel: %s, massage body:
${body}", redisChannel))
                .setHeader(CAMEL_REDIS_CHANNEL, constant(redisChannel))
                .setHeader(CAMEL_REDIS_MESSAGE, body())
                .to(String.format("spring-redis://%s:%s?command=PUBLISH&redisTemplate=#%s",
redisHost, redisPort, ManagedCamel.REDIS_TEMPLATE));
    }
}
```

RedisSubscriber

```
public class RedisSubscriber extends RouteBuilder {

    @Value("${redis.host}")
    private String redisHost;
    @Value("${redis.port}")
```

```
    private int redisPort;
    @Value("${redis.channel.mychannel}")
    private String redisChannel;

    private Object bean;
    private String method;

    @Required
    public void setBean(Object bean) {
        this.bean = bean;
    }

    @Required
    public void setMethod(String method) {
        this.method = method;
    }

    @Override
    public void configure() throws Exception {
        from(String.format("spring-
redis://%s:%s?command=SUBSCRIBE&channels=%s&serializer=#%s", redisHost, redisPort,
redisChannel, ManagedCamel.REDIS_SERIALIZER))
                .log(String.format("Consuming with redis in channel: %s, massage body:
${body}", redisChannel))
                .process(exchange -> {
                }).bean(bean, String.format("%s(${body})", method));
    }
}
```

El método 'Método' dentro del frijol inyectado manejará los masajes recibidos.

Contexto de primavera del suscriptor

```
<bean id="managedCamel" class="com.pubsub.example.ManagedCamel" >
    <constructor-arg name="routes">
        <list>
            <ref bean="redisSubscriber"/>
        </list>
    </constructor-arg>
</bean>

<bean id="redisSubscriber" class="com.pubSub.example.RedisSubscriber" >
    <property name="bean" ref="myBean"/>
    <property name="method" value="process"/>
</bean>
```

Editor de contexto de primavera

```
<bean id="managedCamel" class="com.pubSub.example.ManagedCamel" >
    <constructor-arg name="routes">
        <list>
            <ref bean="redisPublisher"/>
        </list>
    </constructor-arg>
</bean>

<bean id="redisPublisher" class="com.pubSub.example.RedisPublisher" >
```

```
    <property name="producerName" value="direct:myprocedure"/>
</bean>
```

ManagedCamel

```
public class ManagedCamel implements Managed {

    public static final String REDIS_TEMPLATE = "redisTemplate";
    public static final String LISTENER_CONTAINER = "listenerContainer";
    public static final String REDIS_SERIALIZER = "redisSerializer";
    private DefaultCamelContext camelContext;

    private List<RouteBuilder> routes;
    @Value("${redis.host}")
    private String redisHost;
    @Value("${redis.port}")
    private int redisPort;
    @Value("${redis.password}")
    private String redisPassword;

    public ManagedCamel(List<RouteBuilder> routes) throws Exception {
        this.routes = routes;
    }

    @PostConstruct
    private void postInit() throws Exception {
        JndiRegistry registry = new JndiRegistry();
        final StringRedisSerializer serializer = new StringRedisSerializer();
        RedisTemplate<String, Object> redisTemplate = getRedisTemplate(serializer);
        registry.bind(REDIS_TEMPLATE, redisTemplate);
        RedisMessageListenerContainer messageListenerContainer = new
RedisMessageListenerContainer();
        registry.bind(LISTENER_CONTAINER, messageListenerContainer);
        registry.bind(REDIS_SERIALIZER, serializer);

        camelContext = new DefaultCamelContext(registry);
        for (RouteBuilder routeBuilder : routes) {
            camelContext.addRoutes(routeBuilder);
        }
        start();
    }

    private RedisTemplate<String, Object> getRedisTemplate(StringRedisSerializer serializer) {
        RedisTemplate<String, Object> redisTemplate = new RedisTemplate<String, Object>();
        redisTemplate.setConnectionFactory(redisConnectionFactory());
        redisTemplate.setKeySerializer(new StringRedisSerializer());
        redisTemplate.setValueSerializer(serializer);
        redisTemplate.setEnableDefaultSerializer(false);
        redisTemplate.afterPropertiesSet();
        return redisTemplate;
    }

    private RedisConnectionFactory redisConnectionFactory() {
        final JedisConnectionFactory jedisConnectionFactory = new JedisConnectionFactory();
        jedisConnectionFactory.setHostName(redisHost);
        jedisConnectionFactory.setPort(redisPort);
        jedisConnectionFactory.setPassword(redisPassword);
        jedisConnectionFactory.afterPropertiesSet();
        return jedisConnectionFactory;
```

```
    }

    public void start() throws Exception {
        camelContext.start();
    }

    public void stop() throws Exception {
        camelContext.stop();
    }

    public ProducerTemplate createProducer() {
        return camelContext.createProducerTemplate();
    }

}
```

Lea Pub / Sub usando Camel + Redis en línea: https://riptutorial.com/es/apache-camel/topic/7105/pub---sub-usando-camel-plus-redis

Creditos

S. No	Capítulos	Contributors
1	Empezando con el camello apache	Community, Michael Hoffman, Namphibian
2	Pruebas de integración en rutas existentes con Apache-Camel y Spring (y DBUnit)	DamienB, Flanfl, matthieusb
3	Pub / Sub usando Camel + Redis	Lior