Бесплатная электронная книга

УЧУСЬ apache-camel

Free unaffiliated eBook created from
Stack Overflow contributors.

#apache-camel

1
1: apache-camel 2
2
Examples 2
2
Maven 2
Gradle 3
3
3
2: / Camel + Redis 5
5
Examples 5
RedisPublisher 5
RedisSubscriber 5
6
6
ManagedCamel 7
3: Apache-Camel Spring (9
9
9
10
Examples 10
10
Camel 11
Camel Integration test 12
16

Около

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

глава 1: Начало работы с apache-camel

замечания

Apache Camel - это основа, которая в первую очередь облегчает решение проблем интеграции предприятий. По своей сути это можно рассматривать как конструктор движка двигателя маршрутизации. По сути, он позволяет подключать системы (конечные точки) по маршрутам. Эти маршруты принимают сообщения, которые могут быть любого типа данных.

Рамка Apache Camel также содержит полный набор шаблонов интеграции EIP (например, сплиттер, агрегаторы, маршрутизацию на основе контента и т. Д.). Поскольку инфраструктура может быть развернута в различных автономных приложениях Java, на различных серверах приложений, таких как WildFly и Tomcat, или на полноценной корпоративной служебной шине, ее можно рассматривать как интеграционную структуру.

Чтобы начать работу с каркасом, вам нужно добавить его в проект, используя один из следующих способов:

1. специалист
2. Gradle
3. Весенняя загрузка
4. В проект добавлена простая старая справочная библиотека JAR.

Examples

Установка или настройка

Подробные инструкции по добавлению необходимых зависимостей Camel.

Зависимость Maven

Одним из наиболее распространенных способов включения Apache Camel в ваше приложение является зависимость от Maven. Добавив блок зависимостей ниже, Maven разрешит вам библиотеки и зависимости Camel.

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

Gradle

Другим распространенным способом включения Apache Camel в ваше приложение является зависимость от Gradle. Просто добавьте строку зависимостей ниже, и Gradle импортирует библиотеку Camel и ее зависимости для вас.

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

Весенняя загрузка

Начиная с Camel 2.15, теперь вы можете использовать зависимость Spring Boot Apache Camel. Разница с этой библиотекой Camel заключается в том, что она обеспечивает самоуверенную автоконфигурацию, включая автоматическое обнаружение маршрутов 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>
```

Специфический язык домена верблюда

DSL Camel (Domain Specific Language) - одна из особенностей, которая делает превосходство Camel в других рамках интеграции. Хотя некоторые другие структуры также имеют концепцию DSL, как правило, в виде XML-файла, DSL всегда был обычным языком.

Camel предлагает несколько DSL в языках программирования, таких как Java, Scala, Groovy и в XML.

Например, простой путь копирования файлов может быть выполнен различными способами, как показано в списке ниже

- 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"
```

Прочитайте Начало работы с apache-camel онлайн: https://riptutorial.com/ru/apache-camel/topic/3511/начало-работы-с-apache-camel

глава 2: Паб / Суб с помощью Camel + Redis

замечания

Использование издателя:

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

Используя «createProducer ()» в ManagedCamel, создайте sampleTemplate.

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));
    }
}
```

Метод «метод» внутри инъецируемого компонента будет обрабатывать восстановленные массы.

Подлинный контекст подписчика

```
<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>
```

Весенний контекст источника

```
<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();
    }

}
```

Прочитайте Паб / Суб с помощью Camel + Redis онлайн: https://riptutorial.com/ru/apache-camel/topic/7105/паб---суб-с-помощью-camel-plus-redis

глава 3: Тестирование интеграции существующих маршрутов с помощью Apache-Camel и Spring (и DBUnit)

Вступление

Цель этой вики - показать вам, как выполнять интеграционные тесты с помощью Apache Camel.

Точнее, при этом вы сможете запустить существующий маршрут от начала до конца (с или без вашей реальной базы данных) или перехватить обмен между каждой частью маршрута и проверить, правильны ли ваши заголовки или тело.

Проект, который я делал это, использует классическую Spring с конфигурацией xml и DBUnit, чтобы издеваться над тестовой базой данных. Надеюсь, это даст вам несколько результатов.

параметры

Параметр / Функция	подробности
обмен	Обмен используется внутри процессора верблюда для передачи объектов между частями вашего маршрута
CamelContext	Контекст верблюда используется в тесте для ручного запуска и остановки контекста.
ProducerTemplate	Позволяет отправлять сообщения на вашем маршруте, настраивая полный обмен вручную или отправляя фиктивные заголовки / тело
AdviceWith	Помогает вам переопределить существующий маршрут с текущим контекстом
WeaveById	Используется внутри совета с настройкой, сообщает части вашего маршрута, как вести себя (может также использовать *weaveByToString*)
MockEndpoint	Точка макета - это точка, которую вы определяете для своего теста. В вашем weaveById вы можете указать свой маршрут на обычную обработку и перейти в mockEnpoint, а не следовать

Параметр / Функция	подробности
	обычным маршрутам. Таким образом, вы можете проверить количество сообщений, статус обмена ...

замечания

Некоторые определения, приведенные здесь, не совсем точны, но они помогут вам понять приведенный выше код. Вот несколько ссылок для более подробной информации:

- Об использовании *AdviceWith* и *weaveById* (или других способов запуска маршрутов), посмотрите официальную документацию apache-camel: см. Эту ссылку
- Об использовании *ProducerTemplate* см. В официальной документации еще раз: см. Эту ссылку
- Чтобы действительно понять, что такое верблюд: подробная документация по Entreprise Integration Patterns

Этот особый способ тестирования довольно сложно найти, даже при переполнении стека. Это довольно специфично, но не стесняйтесь спрашивать подробности, может быть, я смогу помочь.

Examples

Пример маршрута верблюда

Следующий маршрут имеет простую цель:

- Во-первых, он проверяет, присутствует ли и объект **ImportDocumentProcess** в базе данных и добавляет его как *заголовок обмена*
- Затем он добавляет **ImportDocumentTraitement** (который связан с предыдущим ImportDocumentProcess) в базе данных

Вот код этого маршрута:

```
@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");
    }

}
```

Идентификатор маршрутов не является обязательным, после этого вы также можете использовать строки bean. Однако я думаю, что использование *идентификаторов* можно считать хорошей практикой, если ваши строки маршрутов изменится в будущем.

Пример процессора Camel

В процессоре просто содержатся методы, необходимые для маршрута. Это просто классический Java Bean, содержащий несколько методов. Вы также можете *реализовать Процессор* и переопределить метод *процесса* .

См. Следующий код:

```
@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
}
```

Здесь не так много сказать, за исключением того, что мы используем обмен для передачи объектов из одной части в другую. Это обычно делается в моем проекте, так как мы имеем очень сложные процессы для обработки.

Пример класса Camel Integration test

> Не забудьте добавить поддержку тестирования верблюдов и поддержку теста верблюжьей верблюды в зависимости от вашего проекта. Для пользователей maven см. Следующее:

```
<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>
```

Этот класс будет запускать и запускать тесты на примере маршрута. Эти тесты также используют **DBUnit** для имитации базы данных, хотя вы можете настроить свой контекст для использования реальной или другой подобранной базы данных.

Во-первых, мы используем абстрактный класс, чтобы совместно использовать общие аннотации между каждым классом Test Integration Test, который мы будем использовать позже:

```
@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 {

}
```

Будьте осторожны, чтобы не забыть какую-либо аннотацию, или ваши DAO не будут правильно введены. При этом вы можете безопасно удалить аннотации DBUnit, если вы не хотите использовать базу данных, изображенную в вашей конфигурации контекста.

ВАЖНОЕ `@DirtiesContext(classMode = ClassMode.AFTER_EACH_TEST_METHOD)` Недавно я добавил `@DirtiesContext(classMode = ClassMode.AFTER_EACH_TEST_METHOD)` . Таким образом, *контекст верблюда* перезагружается для каждого теста. Вы действительно можете проверить каждую часть своего маршрута индивидуально. Однако, если вы действительно этого хотите, вам нужно использовать *remove ()* на участках выбранного маршрута, который вы не хотите проходить. Некоторые утверждают, что это не настоящий тест интеграции, и они были бы правы. Но если, как и я, у вас есть большие процессоры, которые вам нужны для рефакторинга, вы можете начать там.

В приведенном ниже коде показано начало тестового класса (см. Ниже приведенные ниже примеры тестов):

```
@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
    }
```

```
}
```

Следующий тест должен инициировать первую часть маршрута и привести его к `mockEndpoint` чтобы мы могли проверить, правильно ли выбран *ImportDocumentProcess* и помещены в заголовки:

```
@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();
}
```

Последний тест запускает весь маршрут:

```
@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);
}
```

Также можно перенаправить текущий маршрут на другой процесс. Но я предпочитаю перенаправление на `mockEndpoint` . Это немного интереснее, потому что вы действительно можете делать промежуточные тесты на вашем обмене и заголовках.

ВАЖНОЕ ПРИМЕЧАНИЕ : В этом примере я использую следующий фрагмент кода для получения маршрутов и использования `adviceWith` их помощью:

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

ОДНАКО Можно получить маршрут по идентификатору, ранее определенному как строка, например:

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

Я очень рекомендую этот метод, он может сэкономить много времени, выясняя, почему, почему ваши тесты терпят неудачу

Прочитайте Тестирование интеграции существующих маршрутов с помощью Apache-Camel и Spring (и DBUnit) онлайн: https://riptutorial.com/ru/apache-camel/topic/10630/тестирование-интеграции-существующих-маршрутов-с-помощью-apache-camel-и-spring--и-dbunit-

кредиты

S. No	Главы	Contributors
1	Начало работы с apache-camel	Community, Michael Hoffman, Namphibian
2	Паб / Суб с помощью Camel + Redis	Lior
3	Тестирование интеграции существующих маршрутов с помощью Apache-Camel и Spring (и DBUnit)	DamienB, Flanfl, matthieusb