REST Example with tests

- Overview
    - REST service
        - geocodeServiceRoute
        - getLocationRoute
        - getLocationInfoRoute
            - Multicast - sample execution
- Tests
- Resources

Overview

Project available on github: https://github.com/jpolitow/Camel-Rest.git

Sample app provides two REST functions:

- /geoservice/location?address={p1} - returns coordinates of specified address
- /geoservice/locationInfo?lat={p1}&lng={p2}&radius={p3} - returns count of schools and gyms in the area

REST service

REST service

```
@Service("geocodeService")
@Path("/geoservice/")
public class GeocodeRestService {

    @GET
    @Path("/location")
    @Produces(MediaType.APPLICATION_JSON)
    public Response getLocation(@QueryParam("address")
                                @NotNull String address) {
        return null;
    }

 @GET
 @Path("/locationinfo")
 @Produces(MediaType.APPLICATION_JSON)
 public Response getLocationInfo(@QueryParam("lat") @DecimalMin("-90")
@DecimalMax("90")
                                 String latitude,
                                 @QueryParam("lng") @DecimalMin("-180")
@DecimalMax("180")
                                 String longitude,
                                 @QueryParam("radius") @Min(1)
                                 int radius) {
     return null;
 }
}
```

geocodeServiceRoute

geocodeServiceRoute is responsible for routing incoming REST requests into proper Camel routes, it is done by using property {header.operationName} which holds name of REST service method matched with request.

For example when invoking url: /geoservice/location?address=gdansk, matched service method will be: public Response getLocation(String address), so {header.operationName} = "getLocation" and invoked endpoint "direct:getLocation" registered on route with id="getLocationRoute".

serviceRoute

```
<route id="geocodeServiceRoute">
    <from uri="cxfrs:bean:rsServer?bindingStyle=SimpleConsumer"/>
    <recipientList>
        <simple>direct:${header.operationName}</simple>
    </recipientList>
</route>
```

getLocationRoute

Connects with https://maps.googleapis.com/maps/api/geocode/json (documentation: https://developers.google.com/maps/documentation/geocoding/intro) which converts addresses (like "1600 Amphitheatre Parkway, Mountain View, CA") into geographic coordinates, then JSON result is unmarshalled into Map<String, Object> and passed to processLocation method of geocodeProcessor bean (bean binding), which produces JSON Response.

```
<route id="getLocationRoute">
    <from uri="direct:getLocation"/>
    <recipientList delimiter="|">

<simple>{{endpoint.google.api.geocoding}}?address=${header.address}&amp;se
nsor=false&amp;key={{googlekey}}&amp;bridgeEndpoint=true</simple>
    </recipientList>
    <unmarshal ref="jackson"/>
    <bean ref="geocodeProcessor" method="processLocation"/>
</route>
```

getLocationInfoRoute

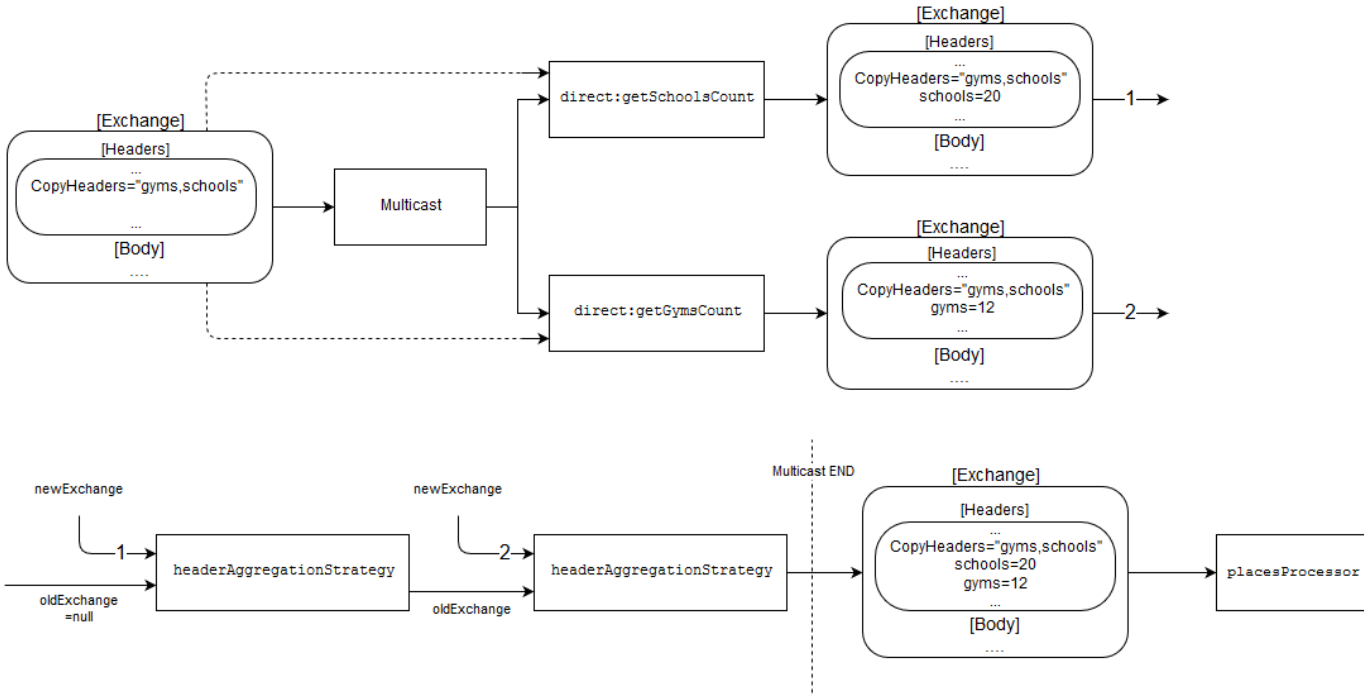
Connects with https://maps.googleapis.com/maps/api/place/radarsearch/json (documentation: https://developers.google.com/places/web-service/search) which allows to search places by parameters like: name, type, location etc. For example purposes we are looking for count of schools and gyms in location from request, google places API allows to provide only one type of place to search, so two requests must be executed. To achieve this we are using multicast component with enabled parallel processing, after that results of requests (header.gyms, header.schools) are passed into method processLocationInfo of placesProcessor bean.

Camel context

```
<route id="getLocationInfoRoute">
    <from uri="direct:getLocationInfo"/>
    <setHeader headerName="CopyHeaders">
        <simple>gyms,schools</simple>
    </setHeader>
    <multicast parallelProcessing="true"
strategyRef="headerAggregationStrategy">
        <to uri="direct:getSchoolsCount"/>
        <to uri="direct:getGymsCount"/>
    </multicast>
    <bean id="processor" ref="placesProcessor"
method="processLocationInfo(${header.gyms}, ${header.schools})"/>
</route>
```

Multicast - sample execution

Diagram below shows example execution flow of getLocationInfoRoute. For this example we assume that direct:getSchoolsCount finished first.



Multicast allows to route the **same** message to a number of endpoints and process them in a different way. So in example above direct:getSchoolsCount and direct:getGymsCount operates on the same Exchange and returns Exchange with new headers (gyms or schools).

Without specifying strategyRef on multicast it uses org.apache.camel.processor.aggregate.UseLatestAggregationStrategy which returns latest Exchange (in our example it will be exchange 2, exchange 1 will be lost).

For this sample I prepared HeaderAggregationStrategy which copies header values specified in header "CopyHeaders" from oldExchange to newExchange and returns newExchange as a result.

Tests

My test assumption was to use the same route definition as in main project. To achieve that some of the endpoints must be replaced with mocks

on runtime. To make it done I've used advice with.

Test example

```
@Test
public void locationTest() throws Exception {

    context.getRouteDefinition("getLocationRoute").adviceWith(context, new
AdviceWithRouteBuilder() {
        @Override
        public void configure() throws Exception {
            //add mock to the end of the route
            weaveAddLast().to("mock:end");

            //replace matching endpoint with mock

weaveByToString(".*https4.*").replace().to("mock:google.api.geocoding");
        }
    });

    // we must manually start when we are done with all the advice with
    context.start();

    MockEndpoint endMock = getMockEndpoint("mock:end");
    MockEndpoint googleApiMock =
getMockEndpoint("mock:google.api.geocoding");

    String json =
IOUtils.toString(this.getClass().getResourceAsStream("/geocodeRestService/
location/geolocation.json"));
    googleApiMock.returnReplyBody(new SimpleBuilder(json));

    Map<String, Object> headers = new HashMap<>();
    headers.put("operationName", "getLocation");

    template.sendBodyAndHeaders("direct:rest.geocodeService", null,
headers);

    endMock.expectedMessageCount(1);
    endMock.assertIsSatisfied();

    Response response =
endMock.getExchanges().get(0).getIn().getBody(Response.class);

    assertEquals(51.5194133,
((Location)response.getEntity()).getLatitude(), 0);
    assertEquals(-0.1269566,
((Location)response.getEntity()).getLongitude(), 0);

    context.stop();
}

@Override
public boolean isUseAdviceWith() {
    return true;
}
```

Resources

- CXFRS Component, JAX-RS : Understanding the Basics
- Multicast, Aggregation Strategy
- Bean binding
- Advice with