Spring Cloud Tutorial- REST Client using Netflix Feign Simple Example

javainuse.com/spring/spring-cloud-netflix-feign-tutorial

Spring Cloud- REST call using Netflix Feign Client

In this post we implement the Netflix Feign client. Previously we had implemented Load Balancing using Netflix Ribbon. The netflix ribbon code here will be the starting point. Will only be making changes in the employee-consumer module by adding the Netflix Feign code. The employee-producer and Eureka Server code will remain the same.

Spring Cloud - Table Of Contents

Microservice Registration and Discovery with Spring cloud using Netflix Eureka- Part 1.
Microservice Registration and Discovery with Spring cloud using Netflix Eureka - Part 2.
Microservice Registration and Discovery with Spring cloud using Netflix Eureka - Part 3.
Microservice Registration and Discovery with Spring cloud using Netflix Eureka - Part 4.
Spring Cloud- Netflix Eureka + Ribbon Simple Example
Spring Cloud- Netflix Eureka + Ribbon + Hystrix Fallback Simple Example
Spring Cloud- Netflix Hystrix Circuit Breaker Simple Example
Spring Cloud- Netflix Feign REST Client Simple Example

What is the Netflix Feign Client? Need for it?

Feign is a java to http client binder inspired by Retrofit, JAXRS-2.0, and WebSocket. Feign's first goal was reducing the complexity of binding Denominator uniformly to http apis regardless of restfulness. Previous examples in the employee-consumer we consumed the REST services exposed by the employee-producer using **REST Template**

But we had to write a lot of code to perform following-

- For Load balancing using Ribbon.
- Getting the Service instance and then the Base URL.
- Make use of the REST Template for consuming service.

The previous code was as below

```
 @Controller
public class ConsumerControllerClient {

 @Autowired
 private LoadBalancerClient loadBalancer;

 public void getEmployee() throws RestClientException, IOException {

  ServiceInstance serviceInstance=loadBalancer.choose("employee-
producer");

  System.out.println(serviceInstance.getUri());

  String baseUrl=serviceInstance.getUri().toString();

  baseUrl=baseUrl+"/employee";

  RestTemplate restTemplate = new RestTemplate();
  ResponseEntity<String> response=null;
  try{
  response=restTemplate.exchange(baseUrl,
    HttpMethod.GET, getHeaders(),String.class);
  }catch (Exception ex)
  {
   System.out.println(ex);
  }
  System.out.println(response.getBody());
 }
```

The previous code, there are chances of exceptions like NullPointer and is not optimal. We will see how the call is made much easier and cleaner using Netflix Feign. If the Netflix Ribbon dependency is also in the classpath, then Feign also takes care of load balancing by default.

Lets Begin-

As mentioned earlier the source code shared in the previous netflix ribbon tutorial will be the starting point. And the employee-consumer model we will be making the changes.

We first add the netflix feign dependency in the pom as follows-

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
 <modelVersion>4.0.0</modelVersion>

 <groupId>com.javainuse</groupId>
 <artifactId>employee-consumer</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <packaging>jar</packaging>

 <parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.4.1.RELEASE</version>
  <relativePath /> <!-- lookup parent from repository -->
 </parent>
 <properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
  <java.version>1.8</java.version>
 </properties>

 <dependencies>
  <dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
  </dependency>

  <dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-eureka</artifactId>
  </dependency>

  <dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-ribbon</artifactId>
  </dependency>

  <dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-feign</artifactId>
  </dependency>


 </dependencies>

 <dependencyManagement>
  <dependencies>
   <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-dependencies</artifactId>
    <version>Camden.SR6</version>
    <type>pom</type>
    <scope>import</scope>
   </dependency>
  </dependencies>
 </dependencyManagement>


</project>
```

We next define a Feign Client by creating an interface with @FeignClient annotation. We also specify the name value as "employee-producer". This value is the name of the service registered using Eureka for discovery. We define the method call to be made to consume the REST service exposed by the employee-producer module.

```
package com.javainuse.services;

import org.springframework.cloud.netflix.feign.FeignClient;
import
org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.javainuse.controllers.Employee;

@FeignClient(name="employee-producer")
public interface RemoteCallService {
 @RequestMapping(method=RequestMethod.GET, value="/employee")
 public Employee getData();

}
```

Next we autowire the RemoteCallService in the ConsumerControllerClient class. Then using it make the REST call. Load Balancing is automatically taken care by Feign Client.

```
package com.javainuse.controllers;

import java.io.IOException;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.client.RestClientException;

import com.javainuse.services.RemoteCallService;

@Controller
public class ConsumerControllerClient {

 @Autowired
 private RemoteCallService loadBalancer;

 public void getEmployee() throws RestClientException, IOException
{

  try {
   Employee emp = loadBalancer.getData();
   System.out.println(emp.getEmpId());
  } catch (Exception ex) {
   System.out.println(ex);
  }
 }

}
```

Finally we annotate the Spring Boot Main class with **@EnableFeignClients**.

```
package com.javainuse;

import java.io.IOException;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.feign.EnableFeignClients;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.Bean;
import org.springframework.web.client.RestClientException;

import com.javainuse.controllers.ConsumerControllerClient;

@SpringBootApplication
@EnableFeignClients
public class SpringBootHelloWorldApplication {

 public static void main(String[] args) throws RestClientException, IOException {
  ApplicationContext ctx =
SpringApplication.run(SpringBootHelloWorldApplication.class, args);

  ConsumerControllerClient consumerControllerClient =
ctx.getBean(ConsumerControllerClient.class);
  System.out.println(consumerControllerClient);
  consumerControllerClient.getEmployee();

 }

 @Bean
 public ConsumerControllerClient consumerControllerClient() {
  return new ConsumerControllerClient();
 }
}
```

As we had done in previous posts- Start the following Spring Boot Applications-

- eureka-server
- employee-producer
- employee-consumer

On running the employee-consumer we get the output as follows-