Using Feign REST Client for Service Invocation

In this section, we will start with one of the popular Spring Cloud Component that is **Feign**.

Feign

The Feign is a declarative web service (HTTP client) developed by **Netflix**. Its aim is to simplify the HTTP API clients. It is a Java to HTTP client binder. If you want to use Feign, create an interface, and annotate it. It provides pluggable annotation support, including Feign annotations and JAX-RS annotations.

It is a library for creating REST API clients. It makes web service clients easier. The developers can use declarative annotations to call the REST services instead of writing representative boilerplate code.

Spring Cloud OpenFeign

**Spring Cloud OpenFeign** provides OpenFeign integrations for Spring Boot apps through auto-configuration and binding to the Spring Environment. Without Feign, in Spring Boot application, we use **RestTemplate** to call the User service. To use the Feign, we need to add **spring-cloud-starter-openfeign** dependency in the pom.xml file.

Let’s simplify the previously developed code using Spring Cloud OpenFeign.

In the previous section, one of the things that we had already encountered is when we were developing currency-conversion-service; how difficult it was to call the REST service. There is a lot of manuals that we have to do to call a very simple service. But still we have written a lot of code for it.

When we work with microservices, there will be a lot of calls to other services. We need not to code like the previous one. Feign is a component that solves this problem. Feign makes it easy to invoke other microservices.

The other additional thing that Feign provides is:  it integrates with the **Ribbon**(client-side load balancing framework).

Let's implement the Feign in our project and invoke other microservices using Feign.

**Step 1:** Select **currency-conversion-service**project.

**Step 2:** Open the **pom.xml** and add the **Feign**dependency. Feign inherits from the **Netflix**.

1. **<**dependency**>**
2. **<**groupId**>**org.springframework.cloud**</**groupId**>**
3. **<**artifactId**>**spring-cloud-starter-feign**</**artifactId**>**
4. **<**version**>**1.4.4.RELEASE**</**version**>**
5. **</**dependency**>**

**Step 3:** Once the dependency is added, **enable** the Feign to scan the clients by adding the annotation **@EnableFeignClients**in the**CurrencyConversionServiceApplication.java**file.

**Step 4:**Define an attribute in the **@EnableFeignClients**annotation. The attribute is the name of the package that we want to scan.

**CurrencyConversionServiceApplication.java**

1. **package** com.javatpoint.microservices.currencyconversionservice;
2. **import** org.springframework.boot.SpringApplication;
3. **import** org.springframework.boot.autoconfigure.SpringBootApplication;
4. **import** org.springframework.cloud.openfeign.EnableFeignClients;
5. @SpringBootApplication
6. @EnableFeignClients("com.javatpoint.microservices.currencyconversionservice")
7. **public** **class** CurrencyConversionServiceApplication
8. {
9. **public** **static** **void** main(String[] args)
10. {
11. SpringApplication.run(CurrencyConversionServiceApplication.**class**, args);
12. }
13. }

We have enabled the Feign in our project. Now, we will use the Feign to invoke the service.

**Step 5:**Create a **Feign proxy** that enables us to talk to external microservices. Let’s create an interface with the name **CurrencyExchangeServiceProxy.**

**Step 6:**Add an annotation **@FeignClient.**Pass the attributes **name** and **URL**.

In the **name** attribute, write the name of the service that we are going to consume. In our case, we are going to consume **currency-exchange-service**. In the **URL** attribute, write the port on which the currency-exchange-service is running.

1. @FeignClient(name="currency-exchange-service", url="localhost:8000")

**Step 7:** Now, we need to define a method that talks to the **currency-exchange-controller**. Open the **Currency-ConverterController.java**file. Copy the **currency-converter** mapping and paste it in the same file.

**Step 8:** Change the mapping name to **/currency-converter-feign/from/{from}/to/{to}/quantity/{quantity}** and the method name to **convertCurrencyFeign.**

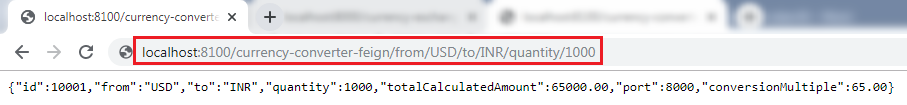
**Step 9:**Make the use of **CurrencyExchangeServiceProxy**and autowired it.

1. @Autowired
2. **private** CurrencyExchangeServiceProxy proxy;

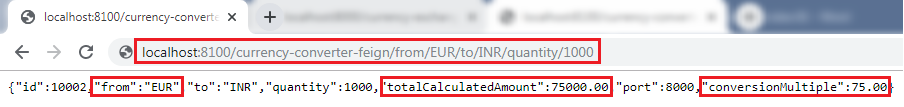
**Step 10:** First, run the **currency-exchange-service** by invoking the URL http://localhost:8000/currency-exchange/from/USD/to/INR after that run the **currency-conversion-service** by using the URL http://localhost:8100/currency-converter/from/USD/to/INR/quantity/1000.

If we do not run the services in the order, the currency-conversion-service shows **Whitelabel Error Page**. It is because the currency-conversion-service uses the conversionMultiple of currency-exchange-service.

**Step 11:**Execute the feign service by using the URL http://localhost:8100/currency-converter-feign/from/USD/to/INR/quantity/1000. It returns the same response as currency-converter-service.



Change the currency **USD** to **EUR** in the above URL and again invoke the same URL. It takes the variable **"from"** from the currency-exchange-service and returns the **totalCalculated Amount.**



The request we are sending uses **Feign**. Feign is a REST Service client. Feign can call the RESTful web services easily. When we use the RestTemplate to call the RESTful service, it creates **duplication** of code that talks to RESTful services.

When we define Feign, we need only to define a proxy and define a single method into it. Feign helps us to simplify client code to talk to the RESTful web services.

Consider a scenario in which currency-exchange-service offers fifteen different services. All the details related to these services must be defined in one place that is **CurrencyExchangeServiceProxy**interface.