IN28MINUTES

Microservices with Spring Boot - Part 4 - Using Ribbon for Load Balancing

Let's learn the basics of microservices and microservices architectures. We will also start looking at a basic implementation of a microservice with Spring Boot. We will create a couple of microservices and get them to talk to each other using Eureka Naming Server and Ribbon for Client Side Load Balancing.

Here is the Microservice Series Outline: Microservices with Spring Boot

- Part 1 - Getting Started with Microservices Architecture
- Part 2 - Creating Forex Microservice
- Part 3 - Creating Currency Conversion Microservice
- Current Part - Part 4 - Using Ribbon for Load Balancing
- Part 5 - Using Eureka Naming Server

This is part 4 of this series. In this part, we will focus on using Ribbon for Load Balancing.

You will learn

- What is the need for Load Balancing?
- What is Ribbon?
- How do you add Ribbon to your Spring Boot Project?
- How do you enable and configure Ribbon to do Load Balancing?

10 Step Reference Courses

- Spring Framework for Beginners in 10 Steps
- Spring Boot for Beginners in 10 Steps
- Spring MVC in 10 Steps
- JPA and Hibernate in 10 Steps
- Eclipse Tutorial for Beginners in 5 Steps
- Maven Tutorial for Beginners in 5 Steps
- JUnit Tutorial for Beginners in 5 Steps
- Mockito Tutorial for Beginners in 5 Steps
- Complete in28Minutes Course Guide

Microservices Overview

In the previous two parts, we created the microservices and established communication between them.

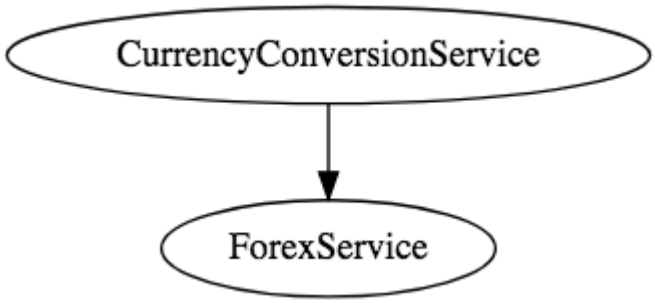
GET to http://localhost:8100/currency-converter-feign/from/EUR/to/INR/quantity/10000

```
{
  id: 10002,
  from: "EUR",
  to: "INR",
  conversionMultiple: 75,
  quantity: 10000,
  totalCalculatedAmount: 750000,
  port: 8000,
}
```

> *When we execute the above service, you would see that a request is also sent over to the forex-service.*

Thats cool!

We have now created two microservices and established communication between them.



However, we are hardcoding the url for FS in CCS component `CurrencyExchangeServiceProxy`.

```
@FeignClient(name="forex-service" url="localhost:8000")
public interface CurrencyExchangeServiceProxy {
  @GetMapping("/currency-exchange/from/{from}/to/{to}")
  public CurrencyConversionBean retrieveExchangeValue
    (@PathVariable("from") String from, @PathVariable("to") String to)
}
```

That means when new instances of Forex Service are launched up, we have no way to distributing load to them.

In this part, let's now enable client side load distribution using Ribbon.

Tools you will need

- Maven 3.0+ is your build tool
- Your favorite IDE. We use Eclipse.
- JDK 1.8+

Complete Maven Project With Code Examples

> *Our Github repository has all the code examples - https://github.com/in28minutes/spring-boot-examples/tree/master/spring-boot-basic-microservice*

Enabling Ribbon

Add Ribbon Dependency to pom.xml

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

Enable RibbonClient in CurrencyExchangeServiceProxy

```
@FeignClient(name="forex-service")
@RibbonClient(name="forex-service")
public interface CurrencyExchangeServiceProxy {
```

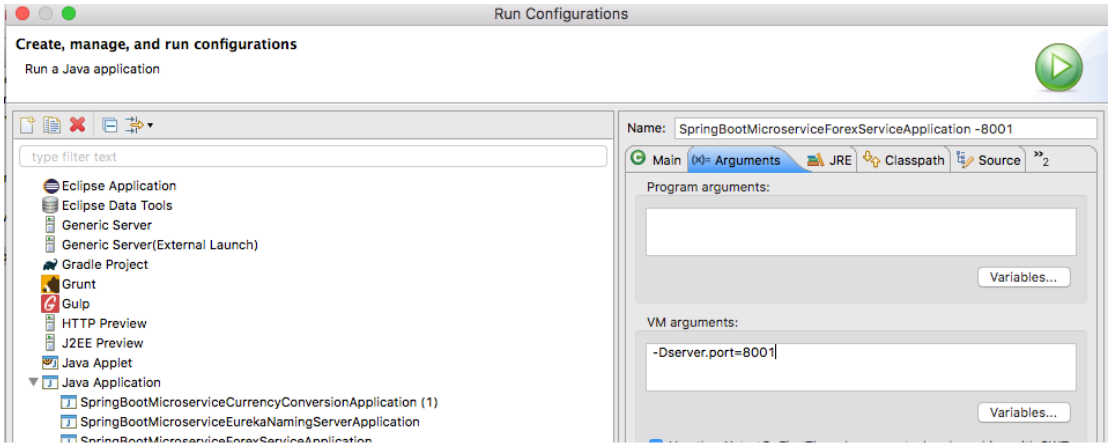
Configure the instances in application.properties

```
forex-service.ribbon.listOfServers=localhost:8000,localhost:8001
```

Launch up Forex Service on 8001

In the above step, we configured ribbon to distribute load to instances. However, we do not have any instance of Forex Service running on 8001.

We can launch it up by configuring a launch configuration as shown in the figure below.



Ribbon in Action

Currently we have the following service up and running

- Currency Conversion Micro Service (CCS) on 8100
- Two instances of Forex MicroService on 8000 and 8001

Now you would see that the requests to CCS would get distributed between the two instances of Forex Microservice by Ribbon

Request 1

GET to http://localhost:8100/currency-converter-feign/from/EUR/to/INR/quantity/10000


```
{
  id: 10002,
  from: "EUR",
  to: "INR",
  conversionMultiple: 75,
  quantity: 10000,
  totalCalculatedAmount: 750000,
  port: 8000,
}
```


Request 2

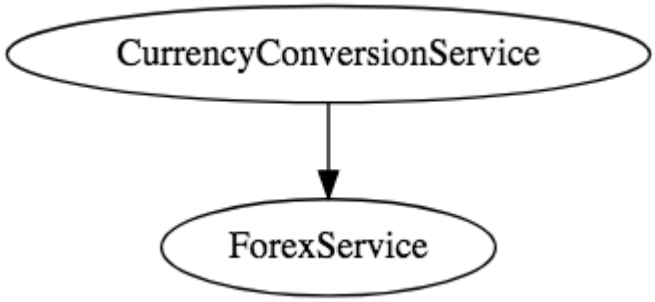
GET to http://localhost:8100/currency-converter-feign/from/EUR/to/INR/quantity/10000


```
{
  id: 10002,
  from: "EUR",
  to: "INR",
  conversionMultiple: 75,
  quantity: 10000,
  totalCalculatedAmount: 750000,
  port: 8001,
}
```


You can see that the port numbers in the two responses are different.

Summary

We have now created two microservices and established communication between them.



We are using Ribbon to distribute load between the two instances of Forex Service.

However, we are hardcoding the urls of both instances of FS in CCS. That means every time there is a new instance of FS, we would need to change the configuration of CCS. Thats not cool.

In the next part, we will use Eureka Naming Server to fix this problem.

Next Steps

- Join 100,000 Learners and Become a Spring Boot Expert - 5 Awesome Courses on Microservices, API's, Web Services with Spring and Spring Boot. Start Learning Now
- Learn Basics of Spring Boot - Spring Boot vs Spring vs Spring MVC, Auto Configuration, Spring Boot Starter Projects, Spring Boot Starter Parent, Spring Boot Initializr
- Learn RESTful and SOAP Web Services with Spring Boot
- Learn Microservices with Spring Boot and Spring Cloud
- Watch Spring Framework Interview Guide - 200+ Questions & Answers

> *Subscribe to get amazing offers on all our courses.*

> *Find out how in28Minutes reached 100,000 Learners on Udemy in 2 years. The in28minutes Way - Our approach to creating awesome learning experiences.*