IN28MINUTES

Microservices with Spring Boot - Part 3 - Creating Currency Conversion Microservice

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
- Current Part - Part 3 - Creating Currency Conversion Microservice
- Part 4 - Using Ribbon for Load Balancing
- Part 5 - Using Eureka Naming Server

This is part 3 of this series. In this part, we will focus on creating the Currency Conversion Microservice.

You will learn

- How to create a microservice with Spring Boot?
- How to use RestTemplate to execute a REST Service?
- How to use Feign to execute a REST Service?
- What are the advantages of Feign over RestTemplate?

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

Resources Overview

Currency Conversion Service (CCS) can convert a bucket of currencies into another currency. It uses the Forex Service to get current currency exchange values. CCS is the Service Consumer.

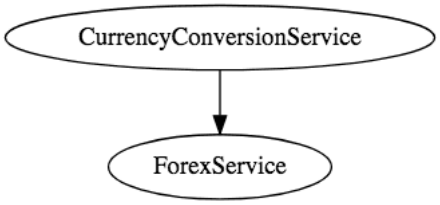
An example request and response is shown below:

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

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

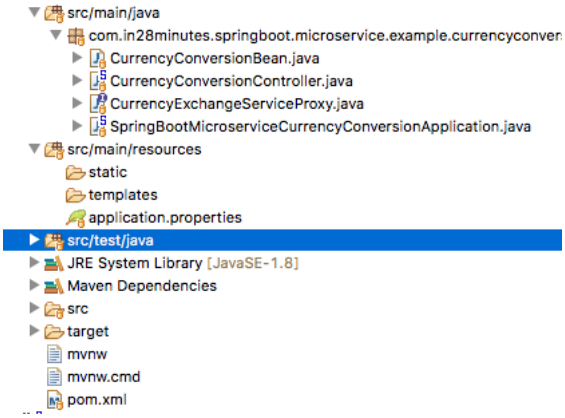
The request above is to find the value of 10000 EUR in INR. The totalCalculatedAmount is 750000 INR.

The diagram below shows the communication between CCS and FS.



Project Code Structure

IN28MINUTES



A few details:

- `SpringBootMicroserviceCurrencyConversionApplication.java` - The Spring Boot Application class generated with Spring Initializer. This class acts as the launching point for application.
- `pom.xml` - Contains all the dependencies needed to build this project. We will use Spring Boot Starter Web.
- `CurrencyConversionBean.java` - Bean to hold the response that we want to send out.
- `CurrencyExchangeServiceProxy.java` - This will be the Feign Proxy to call the Forex Service.
- `CurrencyConversionController.java` - Spring Rest Controller exposing the currency conversion service. This will use the `CurrencyExchangeServiceProxy` to call the Forex Service.

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*

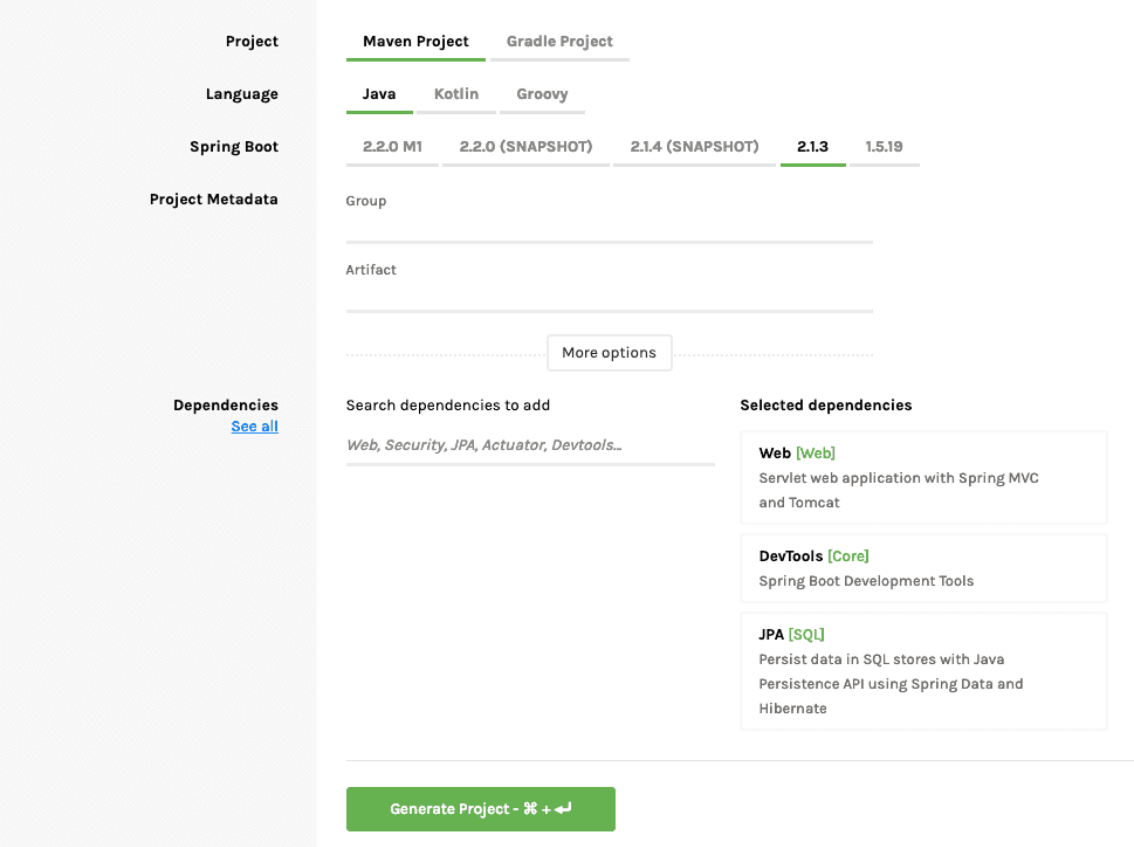
Bootstrapping with Spring Initializr

Creating a Microservice with Spring Initializr is a cake walk.

Spring Initializr http://start.spring.io/ is great tool to bootstrap your Spring Boot projects.

You can create a wide variety of projects using Spring Initializr.

IN28MINUTES



Following steps have to be done for a Web Services project

- Launch Spring Initializr and choose the following
  - Choose `com.in28minutes.springboot.microservice.example.currencyconversion` as Group
  - Choose `spring-boot-microservice-currency-conversion` as Artifact
  - Choose following dependencies
    - Web
    - DevTools
    - Feign
- Click Generate Project.
- Import the project into Eclipse. File -> Import -> Existing Maven Project.

> *Do not forget to choose Feign in the dependencies*

Creating CurrencyConversionBean

This is a simple bean for creating the response.

```
public class CurrencyConversionBean {
  private Long id;
  private String from;
  private String to;
  private BigDecimal conversionMultiple;
  private BigDecimal quantity;
  private BigDecimal totalCalculatedAmount;
  private int port;

  public CurrencyConversionBean() {

  }

  public CurrencyConversionBean(Long id, String from, String to, BigDecimal conversionMultiple, BigDecimal qu
      BigDecimal totalCalculatedAmount, int port) {
    super();
    this.id = id;
    this.from = from;
    this.to = to;
    this.conversionMultiple = conversionMultiple;
    this.quantity = quantity;
    this.totalCalculatedAmount = totalCalculatedAmount;
    this.port = port;
  }
```

Implement REST Client with RestTemplate

The code below shows the implementation of REST Client to call the forex service and process the response. As you can see there is a lot of code that needs to be written for making a simple service call.

```
private Logger logger = LoggerFactory.getLogger(this.getClass());

@GetMapping("/currency-converter/from/{from}/to/{to}/quantity/{quantity}")
public CurrencyConversionBean convertCurrency(@PathVariable String from, @PathVariable String to,
    @PathVariable BigDecimal quantity) {

  Map<String, String> uriVariables = new HashMap<>();
  uriVariables.put("from", from);
  uriVariables.put("to", to);

  ResponseEntity<CurrencyConversionBean> responseEntity = new RestTemplate().getForEntity(
      "http://localhost:8000/currency-exchange/from/{from}/to/{to}", CurrencyConversionBean.class,
      uriVariables);

  CurrencyConversionBean response = responseEntity.getBody();

  return new CurrencyConversionBean(response.getId(), from, to, response.getConversionMultiple(), quantity,
      quantity.multiply(response.getConversionMultiple()), response.getPort());
}
```

Configure application name and port

/spring-boot-microservice-currency-conversion-service/src/main/resources/application.properties

```
spring.application.name=currency-conversion-service
server.port=8100
```

We are assigning an application name as well as a default port of `8100`.

Testing the Microservice

Start the Spring Boot Application by launching SpringBootMicroserviceCurrencyConversionApplication.java

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

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

Creating a Feign Proxy

Feign provide a better alternative to RestTemplate to call REST API.

/spring-boot-microservice-currency-conversion-service/src/main/java/com/in28minutes/springboot/microservice/example/currencyconversion/CurrencyExchangeServiceProxy.java

```
package com.in28minutes.springboot.microservice.example.currencyconversion;

import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.cloud.netflix.ribbon.RibbonClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

@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);
}
```

We first define a simple proxy.

- `@FeignClient(name="forex-service" url="localhost:8100")` - Declares that this is a Feign Client and the url at which forex-service is present is `localhost:8100`
- `@GetMapping("/currency-exchange/from/{from}/to/{to}")` - URI of the service we would want to consume

Using Feign Proxy from the Microservice Controller

Making the call using the proxy is very simple. You can see it in action in the code below. All that we had to do was to autowire the proxy and use to call the method.

```
@Autowired
private CurrencyExchangeServiceProxy proxy;

@GetMapping("/currency-converter-feign/from/{from}/to/{to}/quantity/{quantity}")
public CurrencyConversionBean convertCurrencyFeign(@PathVariable String from, @PathVariable String to,
    @PathVariable BigDecimal quantity) {

  CurrencyConversionBean response = proxy.retrieveExchangeValue(from, to);

  logger.info("{}", response);
```

```
}
```

Enable Feign Clients

Before we are able to use Feign, we need to enable it by using `@EnableFeignClients` annotation on the appropriate package where the client proxies are defined.

```
@SpringBootApplication
@EnableFeignClients("com.in28minutes.springboot.microservice.example.currencyconversion")
@EnableDiscoveryClient
public class SpringBootMicroserviceCurrencyConversionApplication {

  public static void main(String[] args) {
    SpringApplication.run(SpringBootMicroserviceCurrencyConversionApplication.class, args);
  }
}
```

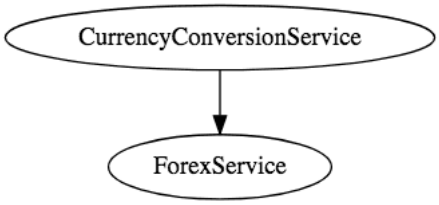
Testing the Microservice using Feign

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,
}
```

Summary

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



However, we are hardcoding the url for FS in CCS. That means when new instances of FS are launched up we have no way to distribute load between them.

In the next part, we will enable client side load distribution using Ribbon.

> *Congratulations! You are reading an article from a series of 50+ articles on Spring, Spring Boot , Hibernate, Full Stack, Cloud and Microservices. We also have 20+ projects on our Github repository. For the complete series of 50+ articles and code examples, click here.*

Next Steps

[![Image]> Congratulations! You are reading an article from a series of 50+ articles on Spring, Spring Boot , Hibernate, Full Stack, Cloud and Microservices. We also have 20+ projects on our Github repository. For the complete series of 50+ articles and code examples, click here.

> ***Join 300,000 Learners!***
>
> *Learn Spring Boot in 10 Steps - FREE Course*

Next Steps

IN28MINUTES

Build Your First Full Stack Application with Angular and Spring Boot. Become a Full Stack Web Developer Now! | By **in28Minutes** Official

4.4 (1,570 ratings)

Docker Crash Course for Java and Spring Boot Developers

HOT & NEW 59 lectures • 6.5 hours • All Levels

Learn Docker containerizing Java Spring Boot Apps - REST API, Full Stack and **Microservices** with Docker Compose | By **in28Minutes** Official

4.6 (44 ratings)

Deploy Spring Boot Apps to Azure with Azure Web App Service

NEW 55 lectures • 6 hours • Beginner

Learn **Azure** deploying Java Spring **REST API**, Full Stack, Docker and Web Apps with **Azure** App Service and **Azure** Web Apps | By **in28Minutes** Official

0.0 (0 ratings)

Deploy Spring Boot Apps to Azure with Azure Web App Service

NEW 55 lectures • 6 hours • Beginner

Learn **Azure** deploying Java Spring **REST API**, Full Stack, Docker and Web Apps with **Azure** App Service and **Azure** Web Apps | By **in28Minutes** Official

0.0 (0 ratings)

Master Microservices with Spring Boot and Spring Cloud

BESTSELLER 127 lectures • 11 hours • All Levels

An awesome journey from Restful Web Services to Microservices with Java, Spring Boot and Spring Cloud | By **in28Minutes** Official

4.4 (10,517 ratings)

Pivotal Cloud Foundry (PCF) Crash Course - Spring Boot Apps

HOT & NEW 58 lectures • 5.5 hours • All Levels

Learn PCF Deploying Java **Spring** Boot REST API, Full Stack Applications and Microservices to Pivotal Cloud Foundry | By **in28Minutes** Official

4.5 (49 ratings)

Deploy Spring Boot Microservices to AWS - ECS & AWS Fargate

89 lectures • 8 hours • All Levels

Learn Amazon Web Services (AWS) and AWS ECS deploying Docker based Spring Boot **Microservices** to AWS Fargate | By **in28Minutes** Official

4.4 (56 ratings)

Deploy Java Spring Boot Apps to AWS with Elastic Beanstalk

BESTSELLER 66 lectures • 6.5 hours • Beginner

Take your first steps towards Amazon Web Services - AWS. Deploy Java Spring Boot REST APIs & Full Stack Apps to AWS. | By **in28Minutes** Official

4.4 (155 ratings)

Master Java Web Services and RESTful API with Spring Boot

BESTSELLER 105 lectures • 9.5 hours • All Levels

Build Amazing Java Web Services - RESTful & SOAP - using Spring & Spring Boot. Master REST APIs & SOAP Web Services Now! | By **in28Minutes** Official

4.4 (3,582 ratings)

Complete Code Example

/spring-boot-microservice-currency-conversion-service/pom.xml

```
<?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.in28minutes.springboot.microservice.example.currency-conversion</groupId>
  <artifactId>spring-boot-microservice-currency-conversion</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>spring-boot-microservice-currency-conversion</name>
  <description>Microservices with Spring Boot and Spring Cloud - Currency Conversion Service</description>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.0.0.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>
    <spring-cloud.version>Finchley.M8</spring-cloud.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-openfeign</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-devtools</artifactId>
      <scope>runtime</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-dependencies</artifactId>
        <version>${spring-cloud.version}</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
```

```
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>

  <repositories>
    <repository>
      <id>spring-snapshots</id>
      <name>Spring Snapshots</name>
      <url>https://repo.spring.io/snapshot</url>
      <snapshots>
        <enabled>true</enabled>
      </snapshots>
    </repository>
    <repository>
      <id>spring-milestones</id>
      <name>Spring Milestones</name>
      <url>https://repo.spring.io/milestone</url>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </repository>
  </repositories>

  <pluginRepositories>
    <pluginRepository>
      <id>spring-snapshots</id>
      <name>Spring Snapshots</name>
      <url>https://repo.spring.io/snapshot</url>
      <snapshots>
        <enabled>true</enabled>
      </snapshots>
    </pluginRepository>
    <pluginRepository>
      <id>spring-milestones</id>
      <name>Spring Milestones</name>
      <url>https://repo.spring.io/milestone</url>
      <snapshots>
        <enabled>false</enabled>
      </snapshots>
    </pluginRepository>
  </pluginRepositories>


</project>
```

/spring-boot-microservice-currency-conversion-service/src/main/java/com/in28minutes/springboot/microservice/example/currencycon

```
package com.in28minutes.springboot.microservice.example.currencyconversion;
import java.math.BigDecimal;

public class CurrencyConversionBean {
  private Long id;
  private String from;
  private String to;
  private BigDecimal conversionMultiple;
  private BigDecimal quantity;
  private BigDecimal totalCalculatedAmount;
  private int port;

  public CurrencyConversionBean() {

  }

  public CurrencyConversionBean(Long id, String from, String to, BigDecimal conversionMultiple, BigDecimal q
      BigDecimal totalCalculatedAmount, int port) {
    super();
    this.id = id;
    this.from = from;
    this.to = to;
    this.conversionMultiple = conversionMultiple;
    this.quantity = quantity;
    this.totalCalculatedAmount = totalCalculatedAmount;
    this.port = port;
  }

  public Long getId() {
    return id;
  }

  public void setId(Long id) {
    this.id = id;
  }

  public String getFrom() {
    return from;
  }

  public void setFrom(String from) {
    this.from = from;
  }
```

```
  }

  public void setTo(String to) {
    this.to = to;
  }

  public BigDecimal getConversionMultiple() {
    return conversionMultiple;
  }

  public void setConversionMultiple(BigDecimal conversionMultiple) {
    this.conversionMultiple = conversionMultiple;
  }

  public BigDecimal getQuantity() {
    return quantity;
  }

  public void setQuantity(BigDecimal quantity) {
    this.quantity = quantity;
  }

  public BigDecimal getTotalCalculatedAmount() {
    return totalCalculatedAmount;
  }

  public void setTotalCalculatedAmount(BigDecimal totalCalculatedAmount) {
    this.totalCalculatedAmount = totalCalculatedAmount;
  }

  public int getPort() {
    return port;
  }

  public void setPort(int port) {
    this.port = port;
  }

}
```

/spring-boot-microservice-currency-conversion-service/src/main/java/com/in28minutes/springboot/microservice/example/currencycon

```
package com.in28minutes.springboot.microservice.example.currencyconversion;

import java.math.BigDecimal;
import java.util.HashMap;
import java.util.Map;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@RestController
public class CurrencyConversionController {

  private Logger logger = LoggerFactory.getLogger(this.getClass());

  @Autowired
  private CurrencyExchangeServiceProxy proxy;

  @GetMapping("/currency-converter/from/{from}/to/{to}/quantity/{quantity}")
  public CurrencyConversionBean convertCurrency(@PathVariable String from, @PathVariable String to,
      @PathVariable BigDecimal quantity) {

    Map<String, String> uriVariables = new HashMap<>();
    uriVariables.put("from", from);
    uriVariables.put("to", to);

    ResponseEntity<CurrencyConversionBean> responseEntity = new RestTemplate().getForEntity(
        "http://localhost:8000/currency-exchange/from/{from}/to/{to}", CurrencyConversionBean.class,
        uriVariables);

    CurrencyConversionBean response = responseEntity.getBody();

    return new CurrencyConversionBean(response.getId(), from, to, response.getConversionMultiple(), quantity,
        quantity.multiply(response.getConversionMultiple()), response.getPort());
  }

  @GetMapping("/currency-converter-feign/from/{from}/to/{to}/quantity/{quantity}")
  public CurrencyConversionBean convertCurrencyFeign(@PathVariable String from, @PathVariable String to,
      @PathVariable BigDecimal quantity) {

    CurrencyConversionBean response = proxy.retrieveExchangeValue(from, to);

    logger.info("{}", response);

    return new CurrencyConversionBean(response.getId(), from, to, response.getConversionMultiple(), quantity,
        quantity.multiply(response.getConversionMultiple()), response.getPort());
```

IN28MINUTES

/spring-boot-microservice-currency-conversion-service/src/main/java/com/in28minutes/springboot/microservice/example/currencycon

```
package com.in28minutes.springboot.microservice.example.currencyconversion;

import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.cloud.netflix.ribbon.RibbonClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

@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);
}
```

/spring-boot-microservice-currency-conversion-service/src/main/java/com/in28minutes/springboot/microservice/example/currencycon

```
package com.in28minutes.springboot.microservice.example.currencyconversion;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.netflix.feign.EnableFeignClients;

@SpringBootApplication
@EnableFeignClients("com.in28minutes.springboot.microservice.example.currencyconversion")
public class SpringBootMicroserviceCurrencyConversionApplication {

  public static void main(String[] args) {
    SpringApplication.run(SpringBootMicroserviceCurrencyConversionApplication.class, args);
  }
}
```

/spring-boot-microservice-currency-conversion-service/src/main/resources/application.properties

```
spring.application.name=currency-conversion-service
server.port=8100
```

/spring-boot-microservice-currency-conversion-service/src/test/java/com/in28minutes/springboot/microservice/example/currencyconv

```
package com.in28minutes.springboot.microservice.example.currencyconversion;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)
@SpringBootTest
public class SpringBootMicroserviceCurrencyConversionApplicationTests {

  @Test
  public void contextLoads() {
  }

}
```