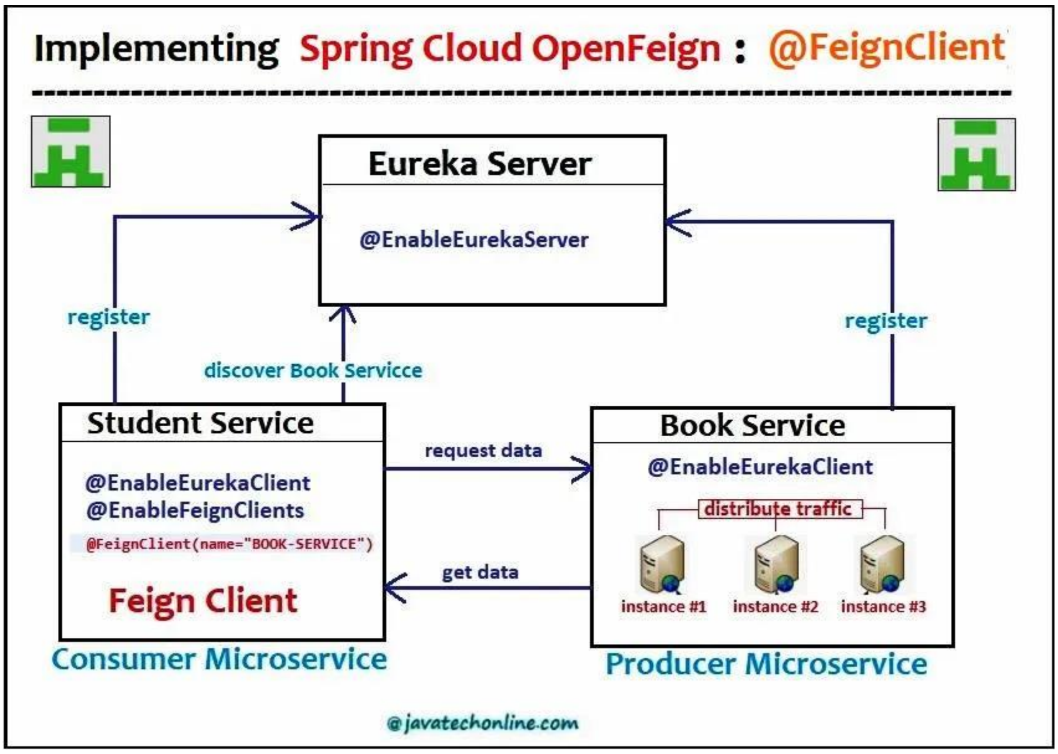
Implement Feign Client In Spring Boot Microservices

When two web applications communicate with each other for data exchange, they work on Producer-Consumer technique. An application who produces data is known as a Producer/Provider application. Similarly the one who consumes data is known as Consumer application. As a Java developer, we might be very familiar with REST API for Producer application whereas RestTemplate for Consumer application. With Microservices based application also, two Microservices communicate with each other and follow the Producer-Consumer model. Here, in consumer side, we use a concept 'Feign Client' as a better option instead of RestTemplate in order to minimize our effort of coding. Therefore, our topic of discussion is 'How to Implement Feign Client in Spring Boot Microservices?'.

Apart from consuming REST services in an easy way, FeignClient/OpenFeign when combined with Eureka also offers us an easy load balancing. We will discuss about both the features of OpenFeign step by step in this article. Let's discuss our topic 'How to Implement Feign Client in Spring Boot Microservices?' and the related concepts.



Create Microservice #1(Eureka Server)

In order to discover and communicate Microservices with each other, we need to create a Eureka Server Service. Creating a Eureka Server is itself similar to creating a Microservice. Moreover, it is just a Spring Boot Project that incorporates Spring Cloud's Eureka Server dependency. In 'application. properties' file we will have some specific properties that will indicate that this application/microservice is a Eureka server. In order to create Eureka Server follow the below steps.

Step #1: Create a Spring Boot Project

Here, we will use STS(Spring Tool Suite) to create our Spring Boot Project. If you are new to Spring Boot, visit Internal Link to create a sample project in spring boot. While creating a project in STS, add starter 'Eureka Server' in order to get features of it.

Step #2: Apply Annotation @EnableEurekaServer at the main class

In order to make your application/microservice acts as Eureka server, you need to apply @EnableEurekaServer at the main class of your application.

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication
@EnableEurekaServer
public class SpringCloudEurekaServerApplication {

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

Step #3: Modify application.properties file

```
server.port=8761
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

eureka.client.register-with-eureka=false

Default value of property 'eureka.client.register-with-eureka' is true. Please note that this property is mandatory to include in Eureka Server in order to make its value as false. However, this is optional to add in case of other microservices/applications that are not Eureka server. Moreover, every microservice project is connected to Spring Cloud project that provides default value to true. Therefore, We should include 'eureka.client.register-with-eureka=false' for one time only in case of Eureka server as Eureka Server itself can't be registered.

eureka.client.fetch-registry=false
Default port for Eureka Server is 8761.

Create Microservice #2(Producer Service)

Step #1: Create a Spring Boot Project

Here, we will use STS(Spring Tool Suite) to create our Spring Boot Project. If you are new to Spring Boot, visit Internal Link to create a sample project in spring boot. While creating a project in STS, add starter 'Eureka Discovery Client' , 'Spring Web' and 'Lombok' in order to get all required features. Furthermore, if you are new to 'Lombok', kindly visit 'How to configure Lombok' and to know all about it in detail.

Step #2: Apply Annotation @EnableEurekaClient at the main class

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;

@SpringBootApplication
@EnableEurekaClient
public class SpringCloudFeignBookServiceApplication {

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

Step #3: Modify application.properties file

```
server.port=9000
spring.application.name=BOOK-SERVICE
eureka.client.service-url.default-zone=http://localhost:8761/eureka        #Register with Eureka
```

Moreover, 'eureka.client.service-url.default-zone=http://localhost:8761/eureka' indicates that this service is getting registered with the Eureka Server.

Step #4: Create Model class as Book.java

```
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
```

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Book {

    private Integer bookId;
    private String bookName;
    private Double bookCost;
}
```

Step #5: Create a RestContoller class as BookRestController.java

In order to publish Book Service, we will create a BookRestController and define the endpoints as below. Please note that, here we are not using a database as our focus is on Feign Client functionality. Instead, we are using some hardcoded values and the Collections to store values wherever required.

```java
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.env.Environment;
import org.springframework.http.HttpStatus;
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.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.dev.springcloud.feign.model.Book;

@RestController
@RequestMapping("/book")
public class BookRestController {

    @Autowired
    Environment environment;

    @GetMapping("/data")
    public String getBookData() {

      return "data of BOOK-SERVICE, Running on port: "
        +environment.getProperty("local.server.port");
    }

    @GetMapping("/{id}")
    public Book getBookById(@PathVariable Integer id) {
      return new Book(id, "Head First Java", 500.75);
    }

    @GetMapping("/all")
    public List<Book> getAll(){
      return List.of(
          new Book(501, "Head First Java", 439.75),
          new Book(502, "Spring in Action", 340.75),
          new Book(503, "Hibernate in Action", 355.75)
      );
    }

    @GetMapping("/entity")
    public ResponseEntity<String> getEntityData() {
      return new ResponseEntity<String>(
       "Hello from BookRestController",
        HttpStatus.OK);
    }
}
```

Create Microservice #3(Consumer Service)

Step #1: Create a Spring Boot Project

Here, we will use STS(Spring Tool Suite) to create our Spring Boot Project. If you are new to Spring Boot, visit Internal Link to create a sample project in spring boot. While creating a project in STS, add starter 'OpenFeign', 'Eureka Discovery Client' , 'Spring Web' and 'Lombok' in order to get all required features. Furthermore, if you are new to 'Lombok', kindly visit 'How to configure Lombok' and to know all about it in detail.

Step #2: Apply Annotation @EnableEurekaClient and @EnableFeignClients at the main class

In order to get the features of OpenFeign, we will additionally need to apply @EnableFeignClients

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.cloud.openfeign.EnableFeignClients;

@SpringBootApplication
@EnableEurekaClient
@EnableFeignClients
public class SpringCloudFeignStudentServiceApplication {

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

Step #3: Modify application.properties file

Add below properties in your application.properties file.

```
server.port=9100
spring.application.name=STUDENT-SERVICE
```

```
eureka.client.service-url.default-zone=http://localhost:8761/eureka
```

Needless to say, we need to add 'eureka.client.service-url.default-zone=http://localhost:8761/eureka' in order to get Student Service registered with Eureka Server.

Step #4: Create Model class as Book.java

Let's replicate the model class as Book.java as it is in the producer service. Please note that, here we have used 'Lombok' annotations in order to reduce the boilerplate code.

```java
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class Book {

    private Integer bookId;
    private String bookName;
    private Double bookCost;
}
```

Step #5: Create an interface as BookRestConsumer.java

This is the most important file when talking about Feign Client. At Interface level, we need to apply @FeignClient annotation and provide the name of producer service/application as below. Further, declare the methods as per the requirement.

```java
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import com.dev.springcloud.feign.model.Book;

@FeignClient(name="BOOK-SERVICE")
public interface BookRestConsumer {

    @GetMapping("/book/data")
    public String getBookData();

    @GetMapping("/book/{id}")
    public Book getBookById(@PathVariable Integer id);

    @GetMapping("/book/all")
    public List<Book> getAllBooks();
    @GetMapping("/book/entity")
    public ResponseEntity<String> getEntityData();
}
```

Step #6: Create a RestController as StudentRestController.java

At the end, create a RestController as StudentRestController to receive the data from Book service as below. The important point to note here is that we need to auto-wire the BookRestConsumer here in this class and then use it.

```
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.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.dev.springcloud.feign.consumer.BookRestConsumer;

@RestController
@RequestMapping("/student")
public class StudentRestController {

    @Autowired
    private BookRestConsumer consumer;

    @GetMapping("/data")
    public String getStudentInfo() {
      System.out.println(consumer.getClass().getName());  //prints as a proxy class
      return "Accessing from STUDENT-SERVICE ==> " +consumer.getBookData();
    }

    @GetMapping("/allBooks")
    public String getBooksInfo() {
      return "Accessing from STUDENT-SERVICE ==> " + consumer.getAllBooks();
    }

    @GetMapping("/getOneBook/{id}")
    public String getOneBookForStd(@PathVariable Integer id) {
      return "Accessing from STUDENT-SERVICE ==> " + consumer.getBookById(id);
    }

    @GetMapping("/entityData")
    public String printEntityData() {
      ResponseEntity<String> resp = consumer.getEntityData();
      return "Accessing from STUDENT-SERVICE ==> " + resp.getBody() +" , status is:" +
resp.getStatusCode();
    }
}
```

That’s All from coding!

How to test FeignClient Enabled Microservice?

It’s time to test our FeignClient enabled Microservice. Please follow below steps:

1) Start Service/Application containing Eureka Server
2) Start Producer Service/Application (Book)
3) Start Consumer Service/Application (Student)
4) Open a browser window, hit below URLs and observe the results.

http://localhost:9100/student/data
http://localhost:9100/student/allBooks
http://localhost:9100/student/getOneBook/501
http://localhost:9100/student/entityData

How to test FeignClient Enabled Microservice as a Load Balancer?

Now we are going to test load-balancing functionality of FeignClient enabled Microservice. Please follow below steps:

1) Start Service/Application containing Eureka Server
2) Start multiple instances of Producer Service/Application (Book). In order to get it, change the server port in application.properties file, save the file and then start the application. Let’s assume that we need three instances of the application. Port 9000 is already configured, now configure two more 9001 and 9002.
3) Start Consumer Service/Application (Student)
4) Now, open a browser window, hit URL ‘http://localhost:9100/student/data’ which contains server port information. Further, refresh the browser multiple times and observe the port number in the results. It will randomly pick the server from 9000, 9001 and 9002 as shown below.