Spring Cloud OpenFeign Example Tutorial

author: Ramesh Fadatare

SPRING BOOT SPRING CLOUD

In a microservices architecture, services often need to communicate with each other. Spring Cloud OpenFeign provides a declarative way to create REST clients, making it easier and more intuitive. In this step-by-step guide, we'll create two simple e-commerce microservices and demonstrate how to use Spring Cloud OpenFeign to call one service from another.

What is Spring Cloud OpenFeign?

Spring Cloud OpenFeign is a Spring Cloud project that integrates with Feign, a declarative web service client. It simplifies the process of creating REST clients by allowing developers to define an interface and annotate it. The implementation is generated at runtime, reducing boilerplate code.

Key Benefits:

1. **Declarative REST Client**: Define REST clients with interfaces and annotations, reducing boilerplate code.
2. **Integration with Spring Cloud**: Seamlessly integrates with other Spring Cloud components like Eureka for service discovery.
3. **Load Balancing**: Supports client-side load balancing with Spring Cloud LoadBalancer or Netflix Ribbon.
4. **Customizable**: Easily customize requests with Feign's annotations.

We will create two microservices: `ProductService` and `OrderService`. `OrderService` will call `ProductService` to get product details.

1 Create ProductService

Step 1: Using Spring Initializr to create the project:

1. Go to **Spring Initializr**.
2. Set the project metadata:
   - **Group**: `com.example`
   - **Artifact**: `product-service`
   - **Name**: `Product Service`
   - **Description**: `E-commerce Product Service`
   - **Package Name**: `com.example.productservice`
   - **Packaging**: `Jar`
   - **Java**: `11` (or higher)
3. Add dependencies:
   - **Spring Web**
4. Click on **Generate** to download the project.
5. Unzip the downloaded project and open it in your IDE.

Step 2: Define the Product model:

Create a new class `Product` in `src/main/java/com/example/productservice`:

```
package com.example.productservice;

public class Product {
    private Long id;
    private String name;
    private double price;

    // Constructors, getters, and setters
    public Product() {}

    public Product(Long id, String name, double price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }

    public Long getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }
}
```

Step 3: Create a ProductController:

Create a new class `ProductController` in `src/main/java/com/example/productservice`:

```
package com.example.productservice;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
```

```java
@RestController
public class ProductController {

    @GetMapping("/products/{id}")
    public Product getProductById(@PathVariable Long id) {
        // For simplicity, returning a hardcoded product. In a real application, you'd query the database.
        return new Product(id, "Sample Product", 99.99);
    }
}
```

Step 4: Run ProductService:

Spring Initializr has already created the main application class. Ensure it looks like this:

```java
package com.example.productservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

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

2 Create OrderService

Step 1: Using Spring Initializr to create the project:

1. Go to **Spring Initializr**.
2. Set the project metadata:
    - **Group**: `com.example`
    - **Artifact**: `order-service`
    - **Name**: `Order Service`
    - **Description**: `E-commerce Order Service`
    - **Package Name**: `com.example.orderservice`
    - **Packaging**: `Jar`
    - **Java**: `11` (or higher)
3. Add dependencies:
    - **Spring Web**
    - **Spring Cloud OpenFeign**
4. Click on **Generate** to download the project.
5. Unzip the downloaded project and open it in your IDE.

Step 2: Enable Feign Clients in OrderService

Open the `OrderService` project and add the `@EnableFeignClients` annotation to the main application class. This enables Feign client support in the Spring Boot application.

```
package com.example.orderservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.openfeign.EnableFeignClients;

@SpringBootApplication
@EnableFeignClients
public class OrderServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(OrderServiceApplication.class, args);
    }
}
```

Step 3: Define a Feign Client Interface in OrderService

Create a Feign client interface to communicate with `ProductService`.

Create ProductClient interface:

Create a new package `com.example.orderservice.clients` and a new interface `ProductClient` in `src/main/java/com/example/orderservice/clients`:

```
package com.example.orderservice.clients;

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

@FeignClient(name = "product-service", url = "http://localhost:8081")
public interface ProductClient {

    @GetMapping("/products/{id}")
    Product getProductById(@PathVariable("id") Long id);
}

class Product {
    private Long id;
    private String name;
    private double price;

    // Constructors, getters, and setters
    public Product() {}

    public Product(Long id, String name, double price) {
        this.id = id;
        this.name = name;
        this.price = price;
```

```
    }

    public Long getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }
}
```

Explanation:

- `@FeignClient(name = "product-service", url = "http://localhost:8081")`: Defines a Feign client with the name `product-service` that points to `http://localhost:8081`.
- `@GetMapping("/products/{id}")`: Maps to the remote service's endpoint to get product information by ID.
- `Product`: A simple POJO to hold the product information.

Step 4: Use the Feign Client in OrderService

Create a service that uses the Feign client to fetch product information from `ProductService`.

Create OrderService:

Create a new class `OrderService` in `src/main/java/com/example/orderservice`:

```
package com.example.orderservice;

import com.example.orderservice.clients.Product;
import com.example.orderservice.clients.ProductClient;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class OrderService {
```

```
    @Autowired
    private ProductClient productClient;

    public Product getProductById(Long id) {
        return productClient.getProductById(id);
    }
}
```

Step 5: Create a REST Controller in OrderService

Create a REST controller that exposes an endpoint to get product information using the `OrderService`.

Create OrderController:

Create a new class `OrderController` in `src/main/java/com/example/orderservice`:

```
package com.example.orderservice;

import com.example.orderservice.clients.Product;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class OrderController {

    @Autowired
    private OrderService orderService;

    @GetMapping("/orders/{id}/product")
    public Product getProductById(@PathVariable Long id) {
        return orderService.getProductById(id);
    }
}
```

Step 6: Run the Applications

1. **Run `ProductService`:**
    - Ensure `ProductService` runs on port `8081` (default port for this example).
    - Verify it is running by accessing `http://localhost:8081/products/1` in your browser. It should return a sample product.
2. **Run `OrderService`:**
    - Ensure `OrderService` runs on a different port (default port `8080`).
    - Verify it is running by accessing `http://localhost:8080/orders/1/product` in your browser. It should return the product details fetched from `ProductService`.

Conclusion

Spring Cloud OpenFeign simplifies the process of creating REST clients in a microservices architecture. By leveraging declarative REST client definitions, developers can reduce boilerplate code and focus on business logic. With seamless integration with other Spring Cloud components, OpenFeign provides a robust solution for inter-service communication.