Feign Client Spring Boot

explain everything about feign client in relation to spring boot microservices in simple but detail. show how to implement as well. give simple but full code. basically create a cheat sheet

Memory updated

Feign Client in Spring Boot Microservices Cheat Sheet

What is Feign Client?

Feign is a declarative HTTP client in Spring Boot that makes it easy to call other microservices by abstracting the HTTP API calls into simple Java interfaces. It simplifies communication between microservices and eliminates the need for writing REST template boilerplate code.

Key Features:

- **Declarative Client**: Define HTTP clients by just creating interfaces and adding annotations.
- **Load Balancing**: Works with Ribbon, a client-side load balancer (no longer actively maintained, but still used).
- **Retries**: Supports automatic retries and fallbacks (used with Hystrix).
- **Simplifies Code**: No need to manually manage HTTP requests and responses.

Dependencies

To use Feign Client in Spring Boot, add the following dependency in your `pom.xml`:

```xml
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

Enabling Feign in Spring Boot

To enable Feign in your Spring Boot application, add the `@EnableFeignClients` annotation in your main application class:

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

How to Define a Feign Client

To create a Feign client, simply define an interface and annotate it with `` `@FeignClient` ``. This interface will declare methods that correspond to the REST API calls.

Example Feign Client

Let's assume there is another microservice that exposes an endpoint `` `/api/v1/users` `` to get user details.

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

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

    @GetMapping("/api/v1/users/{id}")
    User getUserById(@PathVariable("id") Long id);
}
```

- `` `@FeignClient(name = "user-service", url = "http://localhost:8081")` ``: This defines the Feign client and specifies the URL of the target service. The `` `name` `` is an identifier for this client.
- `` `@GetMapping` ``: Maps to the HTTP endpoint `` `/api/v1/users/{id}` ``.
- `` `User getUserById(Long id)` ``: Declares a method that will automatically invoke the corresponding REST API and return a `` `User` `` object.

How to Use Feign Client in a Service

Inject the Feign client into your service and use it to make API calls.

Example Service

```java
import org.springframework.stereotype.Service;

@Service
public class UserService {

    private final UserClient userClient;

    public UserService(UserClient userClient) {
        this.userClient = userClient;
    }

    public User fetchUserDetails(Long id) {
        return userClient.getUserById(id);
        }
    }
```

- `` `UserClient userClient` ``: The Feign client is injected into the service.
- `` `userClient.getUserById(id)` ``: Calls the Feign client method to get user details.

Full Example Code

1. **UserClient Interface** (Feign client):

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

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

    @GetMapping("/api/v1/users/{id}")
    User getUserById(@PathVariable("id") Long id);
}
```

2. **UserService Class** (Service using Feign client):

```java
import org.springframework.stereotype.Service;

@Service
public class UserService {

    private final UserClient userClient;

    public UserService(UserClient userClient) {
        this.userClient = userClient;
    }

    public User fetchUserDetails(Long id) {
        return userClient.getUserById(id);
    }
}
```

3. **Controller to Trigger API Call**:

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

@RestController
public class UserController {

    private final UserService userService;

    public UserController(UserService userService) {
        this.userService = userService;
    }
```

```
        @GetMapping("/users/{id}")
        public User getUser(@PathVariable Long id) {
            return userService.fetchUserDetails(id);
        }
    }
```

4. **Main Application Class**:

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

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

Exception Handling with Feign

You can also define fallback methods in case of failures by adding a fallback class to handle errors.

Example:

```java
@FeignClient(name = "user-service", url = "http://localhost:8081", fallback =
UserClientFallback.class)
public interface UserClient {
    @GetMapping("/api/v1/users/{id}")
    User getUserById(@PathVariable("id") Long id);
}

@Component
class UserClientFallback implements UserClient {
    @Override
    public User getUserById(Long id) {
        // Return default User or handle error
        return new User(id, "Default", "User");
    }
}
```

Benefits of Using Feign:

1. **Simplifies API Calls**: Reduces boilerplate code when calling external services.
2. **Integration**: Works seamlessly with Spring Cloud features like load balancing and circuit breakers.
3. **Declarative Approach**: Easier to maintain and scale as microservices grow.

This covers the essentials of Feign Client in Spring Boot microservices, along with a full code example.

so give example of how 2 microservices will communicate using feign client

To demonstrate how two microservices communicate using Feign Client, we'll set up two simple microservices: **User Service** and **Order Service**. The **Order Service** will call the **User Service** to fetch user details using Feign Client.

Microservice 1: User Service (Producer)

This service provides user data.

Step 1: Define User Model

```java
public class User {
    private Long id;
    private String name;
    private String email;

    // Constructors, Getters, Setters
}
```

Step 2: Create User Controller

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

@RestController
public class UserController {

    @GetMapping("/api/v1/users/{id}")
    public User getUserById(@PathVariable("id") Long id) {
        // Example user data for demonstration
        return new User(id, "John Doe", "john.doe@example.com");
    }
}
```

Step 3: Main Application Class

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

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

Microservice 2: Order Service (Consumer)

This service communicates with **User Service** using Feign Client to fetch user details for orders.

Step 1: Add Feign Client Dependency

In the `pom.xml` file, include the Feign Client dependency:

```xml
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

Step 2: Enable Feign Client

In the main class, enable Feign with `@EnableFeignClients`:

```java
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 Feign Client for User Service

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

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

    @GetMapping("/api/v1/users/{id}")
    User getUserById(@PathVariable("id") Long id);
}
```

Step 4: Create a User Model (Same as in User Service)

```java
public class User {
    private Long id;
    private String name;
    private String email;

    // Constructors, Getters, Setters
}
```

Step 5: Order Service

This service fetches user information from the **User Service** using the `UserClient` Feign client.

```java
import org.springframework.stereotype.Service;

@Service
public class OrderService {

    private final UserClient userClient;

    public OrderService(UserClient userClient) {
        this.userClient = userClient;
    }

    public Order getOrderWithUserDetails(Long orderId, Long userId) {
        // Example order
        Order order = new Order(orderId, "Sample Product", 2);
        // Fetch user details from User Service
        User user = userClient.getUserById(userId);
        order.setUser(user);
        return order;
    }
}
```

Step 6: Order Controller

```java
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 {

    private final OrderService orderService;

    public OrderController(OrderService orderService) {
        this.orderService = orderService;
    }

    @GetMapping("/api/v1/orders/{orderId}/users/{userId}")
    public Order getOrderWithUserDetails(@PathVariable Long orderId, @PathVariable Long
userId) {
        return orderService.getOrderWithUserDetails(orderId, userId);
    }
}
```

Step 7: Define Order Model

```java
public class Order {
    private Long orderId;
    private String product;
    private int quantity;
    private User user; // User details will be fetched from User Service

    // Constructors, Getters, Setters
}
```

How It Works:

1. **Order Service** sends a request to **User Service** via the `UserClient` (Feign client) to fetch user details.
2. **User Service** returns user data as JSON, which **Order Service** includes in its response along with the order details.

Communication Flow:

1. A request is made to **Order Service**: `GET /api/v1/orders/1/users/100`
2. **Order Service** uses **Feign Client** to call **User Service**: `GET /api/v1/users/100`
3. **User Service** returns the user details: `{id: 100, name: "John Doe", email: "john.doe@example.com"}`
4. **Order Service** combines the order data with the fetched user data and returns the full response.

Example Request to Order Service

```bash
curl http://localhost:8082/api/v1/orders/1/users/100
```

Response from Order Service

```json
{
  "orderId": 1,
  "product": "Sample Product",
  "quantity": 2,
  "user": {
    "id": 100,
    "name": "John Doe",
    "email": "john.doe@example.com"
  }
}
```

Summary

- **User Service** provides user details.

- **Order Service** calls **User Service** using **Feign Client** to fetch user data and combines it with order data.
- Feign simplifies HTTP calls between microservices, making communication easier and more maintainable.