**Simplifying Microservices Communication Using Spring Cloud OpenFeign**

## What Is a Feign Client?

Netflix provides Feign as an abstraction over REST-based calls, by which microservices can communicate with each other, but developers don't have to bother about REST internal details.

## Why We Use Feign Client

Without Feign, In Spring Boot Applications We will be using RestTemplate to call User service. We need to write code somewhat similar to below.

HTTP Client Using RestTemplate

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 | @GetMapping  **public** List<User> **getAllUsers**() {  System.out.println("Calling User Service using Feign Client!!");  RestTemplate restTemplate = **new** RestTemplate();  ResponseEntity<List<User>> response = restTemplate.exchange(  "<http://localhost:8080/user/>",  HttpMethod.GET,  **null**,  **new** ParameterizedTypeReference<List<User>>() {  });  List<User> users = response.getBody();  **return** users;  }  @GetMapping("{id}")  **public** User **getUserById**(@PathVariable("id") **int** id) {  Map<String, String> uriParams = **new** HashMap<String, String>();  uriParams.put("id", String.valueOf(id));  URI uri = UriComponentsBuilder  .fromUriString("[http://localhost:8080/user/{id](http://localhost:8080/user/%7Bid)}")  .buildAndExpand(uriParams)  .toUri();  System.out.println(uri);  RestTemplate restTemplate = **new** RestTemplate();  User forEntity = restTemplate.getForObject(uri, User.class);  **return** forEntity;  }  @PostMapping  **public** ResponseEntity **addUser**(@RequestBody User user) {  System.out.println("Add user");  System.out.println(user.toString());  RestTemplate restTemplate = **new** RestTemplate();  HttpEntity<User> request = **new** HttpEntity<>(user);  ResponseEntity exchange = restTemplate  .exchange("<http://localhost:8080/user/>", HttpMethod.POST, request, String.class);  **return** exchange;  }  @DeleteMapping("{id}")  **public** ResponseEntity **deleteUser**(@PathVariable **int** id) {  System.out.println("delete user");  Map<String, String> params = **new** HashMap<String, String>();  params.put("id", String.valueOf(id));  RestTemplate restTemplate = **new** RestTemplate();  restTemplate.delete("[http://localhost:8080/user/{id](http://localhost:8080/user/%7Bid)}", params);  **return** **new** ResponseEntity("User Deleted successfully", HttpStatus.OK);  } |

### Getting Started With Spring Cloud OpenFeign

To include Feign in project We need to use artifact id spring-cloud-starter-openfeign

openfeign Maven

|  |  |
| --- | --- |
| 1 2 3 4 | <dependency>  <groupId>org.springframework.cloud</groupId>  <artifactId>spring-cloud-starter-openfeign</artifactId> </dependency> |

**EnableFeignClients**

EnableFeignClients

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 | @SpringBootApplication @EnableFeignClients **public** **class** **SpringCloudFeignClientDemoApplication** {   **public** **static** **void** **main**(String[] args) {  SpringApplication.run(SpringCloudFeignClientDemoApplication.class, args);  } } |

Creating Feign Client With Sensible Defaults.

FeignClient

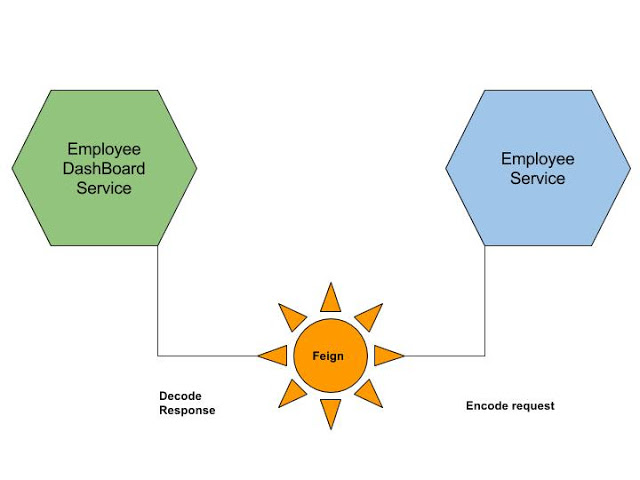
|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 | @FeignClient(name = "User", url = "[http://localhost:8080](http://localhost:8080/)") **public** **interface** **UserClient** {   @RequestMapping(method = RequestMethod.GET, value = "/user")  List<User> **getAllUsers**();   @RequestMapping(method = RequestMethod.GET, value = "/user/{id}")  User **getUser**(@PathVariable("id") **int** id);   @RequestMapping(method = RequestMethod.DELETE, value = "/user/{id}")  ResponseEntity **deleteUser**(@PathVariable("id") **int** id);   @RequestMapping(method = RequestMethod.POST, value = "/user/")  ResponseEntity **addUser**(@RequestBody User user);   @RequestMapping(method = RequestMethod.PUT, value = "/user/")  ResponseEntity **updateUser**(User user);} |

The above code is self explanatory. At minimal we just have to specify name = "User" is an arbitrary client name and url.  
If we compare our UserClient with the code that we have written using RestTemplate, It’s visible that without writing any code specific to calling HTTP Service our UserClient supports all operations. Feign is doing magic under the hood.

## Why We Use Feign Client

In my previous tutorial, When EmployeeDashBoard service communicated with EmployeeService, we programmatically constructed the URL of the dependent microservice, then called the service using RestTemplate, so we need to be aware of the RestTemplate API to communicate with other microservices, which is certainly not part of our business logic.

The question is, why should a developer have to know the details of a REST API? Microservice developers only concentrate on business logic, so Spring addresses this issues and comes with Feign Client, which works on the declarative principle. We have to create an interface/contract, then Spring creates the original implementation on the fly, so a REST-based service call is abstracted from developers. Not only that — if you want to customize the call, like encoding your request or decoding the response in a Custom Object, you can do it with Feign in a declarative way. Feign, as a client, is an important tool for microservice developers to communicate with other microservices via Rest API.

[](https://1.bp.blogspot.com/-cLPct-Cfx3w/WYNgnPfYYJI/AAAAAAAAFqE/3mqTXBrAaOEA9JmkYx9uRgph0prtprSDgCLcBGAs/s1600/Microservices%2BCommunication_%2BFeign%2Bas%2BRest%2BClient.jpg)