MicroService To MicroService communication

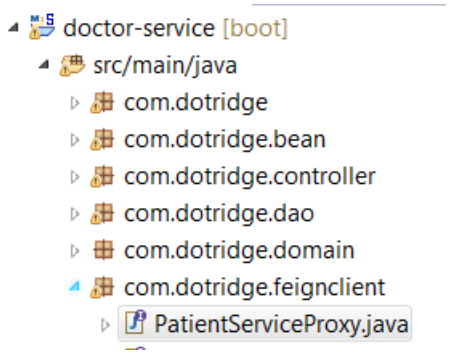
Feign as a declarative REST client

RestTemplate is used for making the synchronous call. When using RestTemplate, the URL parameter is constructed programmatically i.e we have to use the url which typically contains the host name,port context path,resource mapping url like "http://localhost:9090/patient-service/patientApi/createPatient", and data is sent across to the other service. In more complex scenarios, like posting data to another web service we have to get to the details of the HTTP APIs provided by RestTemplate to mentioned the content-type as header parameter and to pass the body with the data at a much lower level. See the below sample:

```
Map<String,String> headers = new HashMap<String,String>();
headers.put("Content-Type", "application/json");
ResponseEntity<PatientBean> resp = restTemplate.postForEntity("http://localhost:9090/patient-service/patientApi/createPatient"
        patientBean, PatientBean.class, headers);
return new ResponseEntity<PatientBean>(resp.getBody(), HttpStatus.OK);
```

Feign is a Spring Cloud Netflix library for **providing a higher level of abstraction over REST-based service calls**. Spring Cloud Feign works on a declarative principle. When using Feign, we write declarative REST service interfaces at the client, and use those interfaces to program the client.

For example: doctor has to register the patient. here **doctor-service** is a micro service should call another micro service **patient-service** and should post the patient data to create the patient.Here **doctor-service** is now acts as a client for **patient-service**, hence we will write a declarative REST service interface I.e.**PatientServiceProxy** at the doctor-service as shown below:

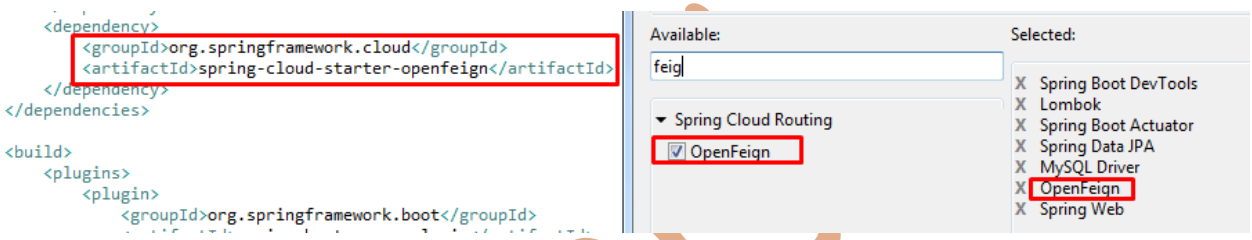


we no need not worry about the implementation of this interface. This will be dynamically provisioned by Spring at runtime. With this declarative approach, we need not get into the details of the HTTP level APIs provided by RestTemplate.

Step-1: In order to use Feign, first we need to change the pom.xml file to include the Feign dependency as follows:

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

For a new Spring Starter project, **Feign** can be selected from the starter library selection screen, or from http://start.spring.io/. This is available under **Cloud Routing** as shown in the following screenshot:



Step-2: Create a PatientServiceProxy interface as below. This should acts as a Proxy interface of the actual **patient-service** micro service.

```
package com.nareshit.proxy;

import org.springframework.cloud.netflix.feign.FeignClient;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;

@FeignClient(name="patient-proxy",url="localhost:9096/patientApi/")
public interface PatientServiceProxy {

    @PostMapping(path = "/createPatient",
            consumes = MediaType.APPLICATION_JSON_VALUE,
            produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<String> addPatient(@RequestBody String payload);
}
```

Note:

If in case the Underlying service required any token header, please pass it as shown in two steps:

Step-1: add @RequestHeader parameter as shown below at proxy:

```java
@FeignClient(name="vehicle-proxy", url="localhost:7075/api/")
public interface VehicleProxy {

    @GetMapping(path="/vehicles")
    public String fetchVehicleById(@RequestHeader Map<String, String> headerMap,
            @PathVariable("vehicleId") String vehicleId);
}
```

Step-2: pass the header value as shown below:

```java
Map<String, String> headers = new HashMap<>();
headers.put("Authorization", "Bearer pPXRsLRxcHnaLdTrQYf0O3afxrBDDR5A");

String vehicleServiceInfo = vproxy.fetchVehicleById(headers,tripBean.getVehicleId());
```

The **PatientServiceProxy** interface has a **@FeignClient** annotation. This annotation tells Spring to create a REST client based on the interface provided. The value could be a service ID or a logical name. The url indicates the actual URL where the target service is running. Either name or value is mandatory. In this case, since we have url, the name attribute is irrelevant. Use this service proxy to call the patient-service micro service. In the doctor-service microservice, we have to tell Spring that Feign clients exist in the Spring Boot application, which are to be scanned and discovered. This will be done by adding @**EnableFeignClients** at the class level of DoctorMServiceApplication. Optionally, we can also give the package names to scan.

```java
@SpringBootApplication
@EnableFeignClients
public class DoctorMServiceApplication  {

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

Step-3: Call patient-service micro service from doctor-micro service:

First autowired the **PatientServiceProxy** in Doctor Controller as shown below:

```
application.properties  application.properties  application.properties  PatientServiceProxy.java  DoctorController.java
package com.dotridge.controller;

import java.util.Date;

@RestController
@RequestMapping("/doctorApi")
public class DoctorController {

    @Autowired
    private DoctorService doctorService;

    /*@Autowired
    private RestTemplate restTemplate;*/

    @Autowired
    private PatientServiceProxy patientProxy;
```

Call patient-service micro service from doctor-micro service as shown below:

```
@PostMapping(path = "/createPatient", consumes = MediaType.APPLICATION_JSON_VALUE, produces = MediaType.APPLICATION_JSON_VALUE)
public ResponseEntity<PatientBean> addPatient(@RequestBody PatientBean patientBean) {
    System.out.println("im in doctor controller add patient");
    patientBean.setCreatedBy("SagarDoctor");
    patientBean.setCreatedDate(ServiceUtil.getStringFromDate(new Date()));

    ResponseEntity<PatientBean> resp = patientProxy.addPatient(patientBean);

    /*Map<String,String> headers = new HashMap<String,String>();
    headers.put("Content-Type", "application/json");
    ResponseEntity<PatientBean> resp = restTemplate.postForEntity(PATIENT_SERVICE, patientBean, PatientBean.class, headers);*/
    // patientBean =restTemplate.getForObject("http://localhost:9090/patient-service/patientApi/createPatient", PatientBean.class);
    return new ResponseEntity<PatientBean>(resp.getBody(), HttpStatus.OK);

}
```

rerun the **doctor-service** microservice to see the effect.The URL of the patient service in the **PatientServiceProxy** interface is hardcoded as **url="localhost:9090/patient-service"**. This is because **@FeignClient** acts as a load balancer, if we forgot to hard code this url, it will try to connect to the load balancer to get the **patient-service** service details, as we didn't configure the load balancer so far, lets hard code this for a time being till we use client side load balancer i.e. Ribbon.

Note:

Value of the name attribute inside the **@FeignClient** annotation I.e.**patient-proxy** is no need to equals to our patient-service micro service name I.e. patient-service.