

Linguagem de Programação Orientada a Objetos 2

Feign

Prof. Tales Viegas

<https://fb.com/ProfessorTalesViegas>

Introdução

- ▶ Em uma arquitetura distribuída, onde uma aplicação deve realizar requisições a outras aplicações através do protocolo HTTP, necessitamos ter uma forma de fazer estas requisições.
- ▶ Em Java SE, podemos usar Sockets ou HTTPClient para realizar estas requisições

Introdução

- ▶ Como o Spring é um framework para aplicações Web, obviamente ele possui uma forma de fazer isto.
- ▶ Por padrão, o Spring possui a classe RestTemplate para realizar requisições a uma determinada URL e mapear o retorno
- ▶ Porém, deve ser construído um RestTemplate para cada uma das URLs que desejamos acessar

Introdução

- ▶ Como já dito, sempre que temos uma tarefa muito difícil de fazer, criamos um Framework para fazer de forma mais fácil
- ▶ Nesta ideia, a Netflix criou o Feign (<https://github.com/OpenFeign/feign>)
- ▶ O Spring incorporou o Feign dentro do seu framework, abaixo do projeto Spring Cloud

Feign

- ▶ Feign é um cliente HTTP Java inspirado nas especificações Retrofit, JAXRS-2.0 e WebSocket.
- ▶ Seu objetivo é reduzir a complexidade da declaração e uso de APIs REST através de chamadas simplificadas, utilizando as mesmas notações das declarações destas APIs

Utilizando Feign

- ▶ Para começar, devemos adicionar a dependência do Feign no projeto

```
implementation('org.springframework.cloud:spring-cloud-starter-openfeign')
```

Utilizando Feign

- ▶ Após, devemos colocar na classe principal do projeto (Application), a notação @EnableFeignClients

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

Utilizando Feign

- Depois basta criarmos uma interface para utilizar o Feign, colocando a notação com a URL do serviço que queremos e os métodos que queremos acessar.

```
@FeignClient(value="party-service", url="$http://localhost:8080")
private interface PartyClient {

    @GetMapping("/v1/party/{partyId}")
    PartyOutput getById(@PathVariable(name = "partyId") Long
                        partyId);
}
```

Utilizando Feign

- ▶ Podemos utilizar, ao invés da URL, uma notação para pegar a URL a partir da configuração do application.yml

```
@FeignClient(value="party-service", url="${url.party-service}")
private interface PartyClient {

    @GetMapping("/v1/party/{partyId}")
    PartyOutput getById(@PathVariable(name = "partyId") Long
                        partyId);
}
```

Utilizando Feign

- ▶ Por fim, basta adicionar a dependência do Feign como um Autowired field no serviço que deseja e chamar os métodos.
- ▶ Caso ocorra qualquer retorno diferente de 200, o Feign gera uma exceção do tipo FeignException

Utilizando Feign

```
@Service
public class PartyClientService {
    private final PartyClient partyClient;

    @Autowired
    public PartyClientService(PartyClient partyClient){
        this.partyClient = partyClient;
    }

    public PartyOutput getById(Long id){
        return this.partyClient.getById(id);
    }
}
```

Material Adicional

- ▶ <https://github.com/OpenFeign/feign>
- ▶ <https://www.baeldung.com/intro-to-feign>
- ▶ https://cloud.spring.io/spring-cloud-netflix/multi/multi_spring-cloud-feign.html
- ▶ <https://domineospring.wordpress.com/2017/06/02/feign-uma-forma-simples-para-consumir-servicos/>