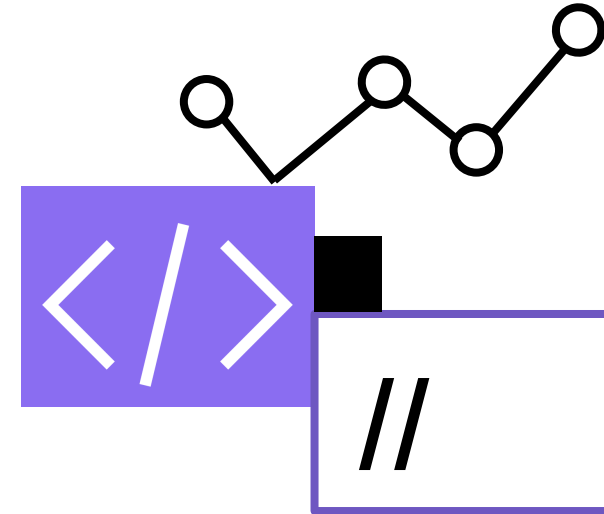


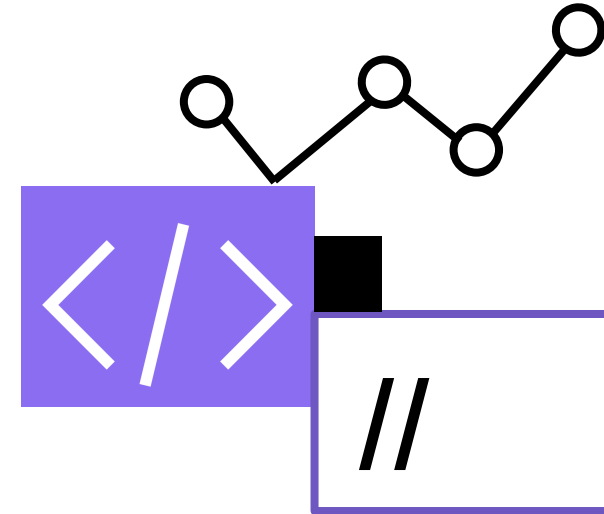
Implementação de Feign Rest Client



Como incluí-lo em nosso projeto?

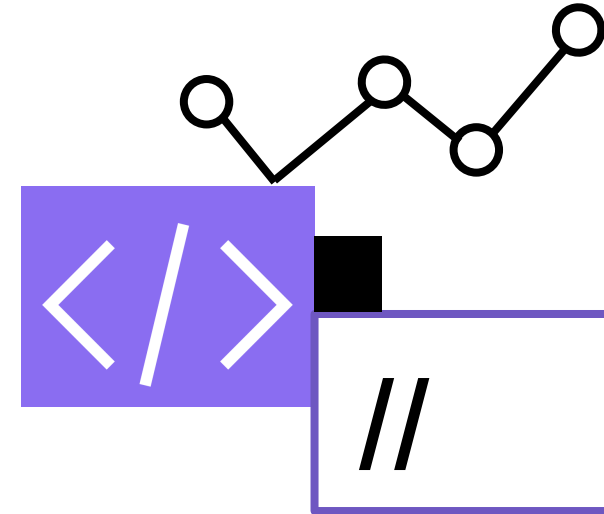
Primeiro, precisamos adicionar a dependência `spring-cloud-starter-openfeign` ao nosso projeto

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



Para habilitar o uso do Feign em nossa aplicação, vamos para a classe principal e adicionamos a anotação `@EnableFeignClients`. Por exemplo:

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



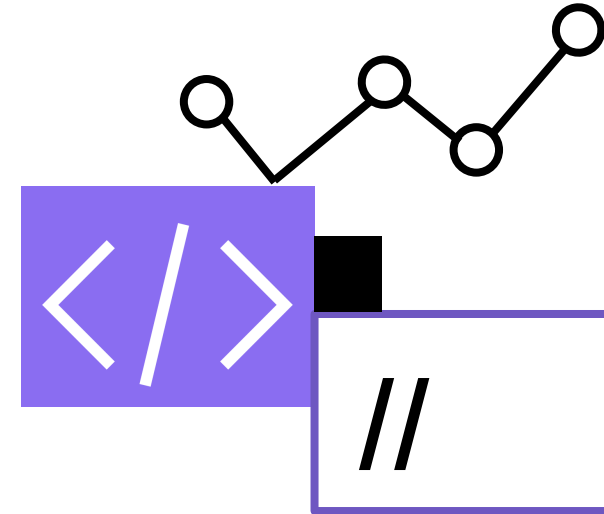
Como enviar solicitações para outro microserviço?

Envio de solicitações entre microserviços registrados no Eureka

Suponha que temos um serviço de produto chamado **product-service**, que expõe uma API REST. Dentro de seus métodos, há um que retorna todos os produtos via GET. A URL é <http://localhost:8080/products>.

Como o microserviço é registrado pelo Eureka, podemos acessar o mesmo recurso substituindo o nome da hostname (localhost) e porta (8080) pelo nome sob o qual ele foi registrado. Em seguida, a URL é <http://product-service/products>. Agora, com Feign usamos interfaces para criar os clientes e enviar os pedidos.

Vejamos um exemplo abaixo.



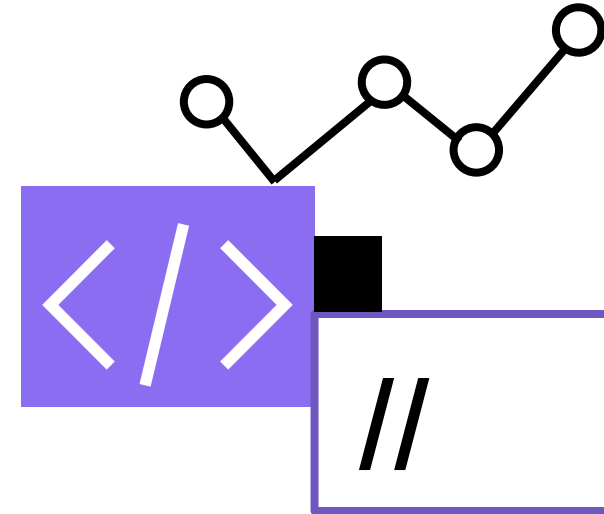
Primeiro, criamos um DTO para armazenar a resposta obtida.

```
public class ProductDTO {  
    private Integer id;  
    private String title;  
    private String color;  
}
```

E então, criamos a interface onde vamos configurar o Feign para enviar a solicitação.

```
@FeignClient(name = "product-service")  
public interface ProductClient {  
    @RequestMapping(method = RequestMethod.GET, value = "/products")  
    List<ProductDTO> getProducts();  
}
```

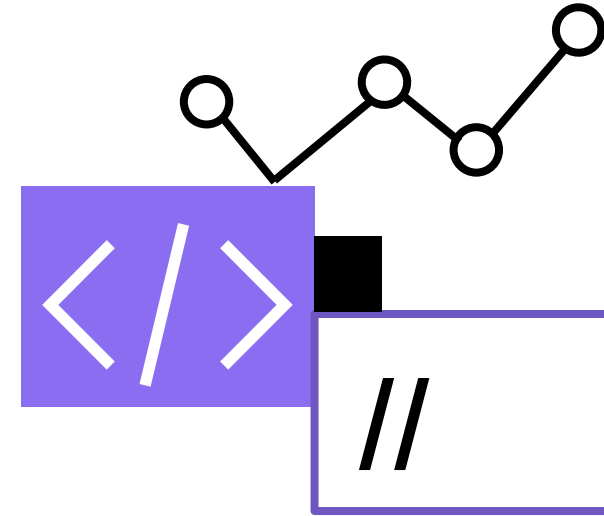
Veremos este pedaço de código em detalhes.



{código}

```
@FeignClient(name = "product-service")
public interface ProductClient {
    @RequestMapping(method = RequestMethod.GET,
value = "/products")
    List<ProductDTO> getProducts();
}
```

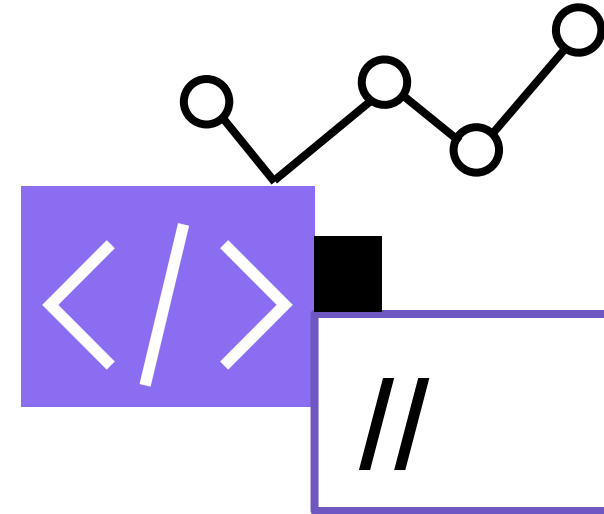
O valor passado na anotação `@FeignClient` é o nome do cliente que estamos criando e também é usado para procurar o endereço físico do microserviço para o qual queremos enviar a solicitação. Ou seja, procure no registro Eureka quais endereços estão associados a esse nome.



{código}

```
@FeignClient(name = "product-service")
public interface ProductClient {
    @RequestMapping(method = RequestMethod.GET,
value = "/products")
    List<ProductDTO> getProducts();
}
```

Com a anotação `@RequestMapping` configuramos o método que vamos usar e a URL. Neste caso, vamos enviar GET a **/products**.



{código}

```
@FeignClient(name = "product-service")
public interface ProductClient {
    @RequestMapping(method = RequestMethod.GET,
value = "/products")
    List<ProductDTO> getProducts();
}
```

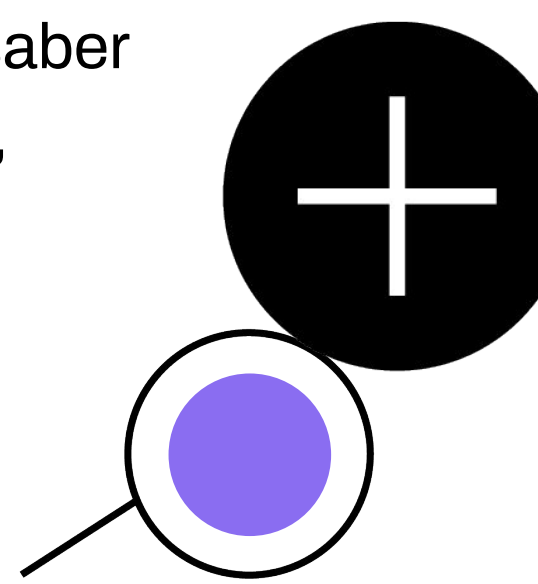
Podemos notar que o método `getProducts()` retorna uma lista de ProductDTO. O Feign tem um decoder por padrão, que analisará o JSON recebido de acordo com a classe que especificamos.

Como enviar solicitações para outro microserviço?

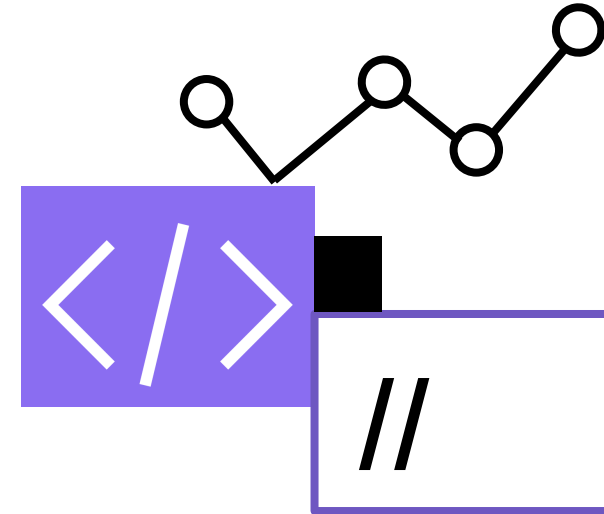
Envio de solicitações a serviços externos

Também podemos enviar solicitações a serviços que são externos ao nosso ecossistema ou que não estão registrados no Eureka. **Para fazer isso, devemos indicar o URL.** Por exemplo, vamos supor que queremos saber de onde no mundo nosso site está sendo visitado. Para resolver isto, fazemos uma solicitação a uma API que, dado um endereço IP, retorna sua localização. Nosso endpoint é <http://ipwhois.app/json/{IP}> e depois configura-se o cliente Feign:

```
@FeignClient(name = "ip-service", url = "http://ipwhois.app/json")
public interface IpClient {
    @RequestMapping(method = RequestMethod.GET, value = "/{ip}")
    List<JsonNode> getStores(@PathVariable("ip") String ip);
}
```



Além de adicionar o parâmetro URL com o endereço, indicamos o nome. Neste caso, ele será utilizado apenas para registrar o cliente no pedido.



Test

Finalmente, para executar os métodos de interface, devemos injetar a interface em um serviço usando a Spring Boot, como mostra o exemplo abaixo:

```
@Service
public class ProductService{

@Autowired
private ProductClient productFeingClient;

public List<ProductDTO> fetchAllProducts(){
    return productFeingClient.getProducts();
}
```

Muito obrigado!