Spring Cloud Feign

Declarative REST Client

Objectives

- At the end of this module, you will be able to
  - Call REST services using the Feign libraries
  - Understand how Feign, Ribbon, and Eureka collaborate

Module Outline

- What is Feign
- How to use Feign

Feign

- What is it?
  - Declarative REST client, from NetFlix
  - Allows you to write calls to REST services with no implementation code
  - Alternative to RestTemplate (even easier!)
  - Spring Cloud provides easy wrapper for using Feign

Spring REST Template

- Spring's Rest Template provides very easy way to call REST services

```
RestTemplate template = new RestTemplate();
String url = "http://inventoryService/{0}";
Sku sku = template.getForObject(url,  Sku.class, 4724352);
```

Instantiate
(or dependency inject)

Provide target URL
(note the placeholder)

Call the URL, provide expected class,
Provide value for placeholder.
Template takes care of all HTTP
and type conversion!

- Still, this code must be
  1) Written
  2) Unit-tested with mocks / stubs.

Feign Alternative –
Declarative Web Service Clients

- How does it work?
  - Define *interfaces* for your REST client code
  - Annotate interface with Feign annotation
  - Annotate methods with Spring MVC annotations
    - Other implementations like JAX/RS pluggable
- Spring Cloud will implement it at run-time
  - Scans for interfaces
  - Automatically implements code to call REST service and process response

Feign Interface

- Create an *Interface*, not a Class:

Marks interface to be implemented by Feign / Spring

```
@FeignClient(url="localhost:8080/warehouse")
public interface InventoryClient {

   @RequestMapping( value = "/inventory", method = RequestMethod.GET )
   List<Item> getItems();

   @RequestMapping( value = "/inventory/{sku}",  method = RequestMethod.POST,
                            consumes = "application/json")
   void update(@PathVariable("sku") Long sku, @RequestBody Item item);
}
```

Base URL
(we'll replace this later with Eureka client ID!)

Ordinary, Spring MVC mapping metadata

- Server can be written in any technology. Not limited to Spring, not limited to Java!
- Note: No extra dependencies are needed for Feign when using Spring Cloud

Runtime Implementations

- Spring scans for @FeignClients
  - Provides implementations at runtime

Spring will seek interfaces and implement them

```
@SpringBootApplication
@EnableFeignClients
public class Application {

    ...

}
```

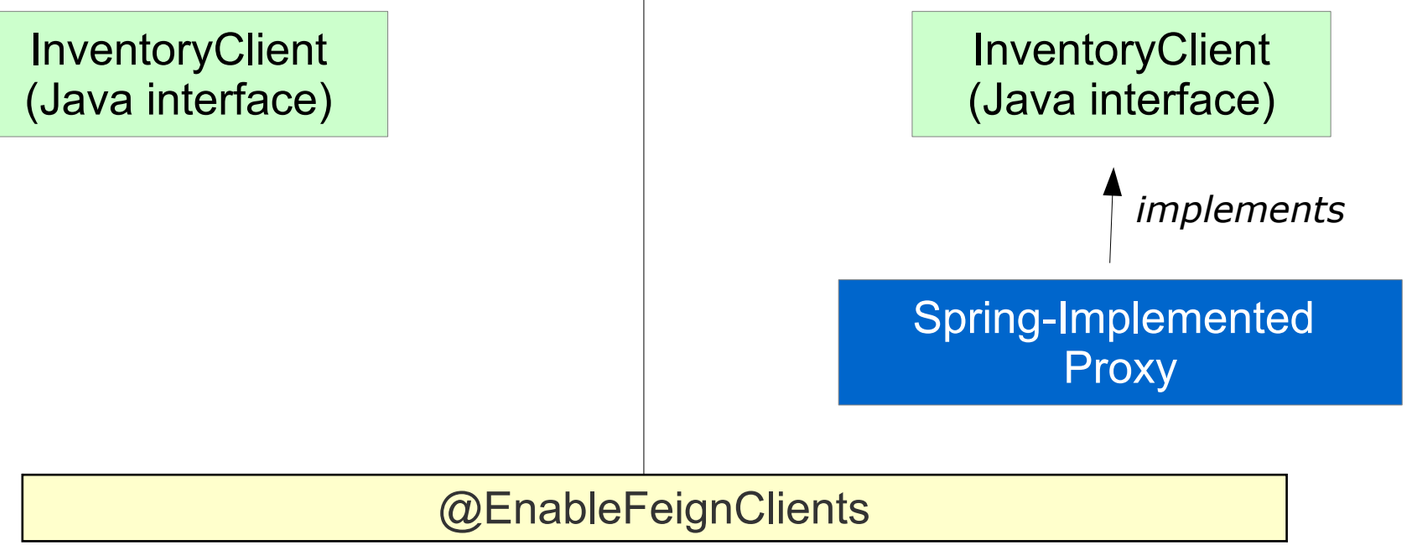
- That's it!
  - Implementations provided by Spring / Feign!

What does @EnableFeignClients do?

- Before startup

InventoryClient
(Java interface)

- After startup



You can @Autowire an InventoryClient wherever one is needed

Ribbon and Eureka

Where do they fit in?

- The previous example - hard-coded URL:

```
@FeignClient(url="localhost:8080/warehouse")
```

- ...use a Eureka “Client ID” instead:

```
@FeignClient("WAREHOUSE")
```

Client's bootstrap.yml:

```
---
spring:
  application:
    name: warehouse
```

- Ribbon is automatically enabled
  - Eureka gives our application all “Clients” that match the given Client ID
  - Ribbon automatically applies load balancing
  - Feign handles the code.

Runtime Dependency

- Feign starter required at runtime:

  ...but not compile time

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

Summary

- Feign provides a very easy way to call RESTful services
- Feign integrates with Ribbon and Eureka automatically.

Exercise

Refactor Client Code to utilize Feign.
Run Eureka, Ribbon, and Feign together.

Instructions: Student Files, Lab 6