More on MicroServices Intra Communicaation

In MicroServices intra communication we need Client Type comps to find the

Service Instance Details of Target Ms and to perform Http calls based communucation with taget MS/Producer MS from Consumer Ms.

Client Type comps are

a) Discovery Client

b) Load BalacerClient (LBC)

c) Fiegn Client (best)

=> if we have 1 producer - 1 consumer type of MicroServices using Feing client then

we need to take 1 Feign Interface (@FeignClient) at Cosumer Ms side... if the

the Conumer Ms is consuming the multiple Producer Ms services then we need to take multiple feign Interfaces.

Feign Interface count = no.of producer Ms services

consumed by ConsumerMs

=> For example if MS#1 is consuming Ms#2 and Ms#3 producer services then Ms#1 consumer Ms should have 2 feign Interfaces as shown below

client

Producer Ms Ms#2

Producer Me Ms#3

Ms

Order Ms (Producer Ms)

Accounts Ms(Producer)

Consumer Ms

MS#1

2 Feign Interfaces

=> 1 for Ms#2 service

=> 1 for Ms#3 service

In this Cosumer Ms

we need to place

2 @FeignClient("....") annotations on

two different feign interfaces having the two different service ids of two different producer Ms. But we need to place @EnableFeignCleints only for 1 time on Main class.

(Cosumer Ms)

Report Generation Ms

=>Contains 1 Feign Interface for Order Ms =>Contains 1 Feign Interface for Accounts Ms

=> if one Producer Ms services are used by multiple consumer Ms then we may need to place same feign Client Interface in multiple consumer Ms.

ProducerMs

Ms#3

Consumer Ms

Consumer Ms

Ms#1

Ms#2

=>contains 1 feign Client interface for Ms#3

=>contains 1 feign Client interface for Ms#3

(Both may be same interfaes)

Inventory Ms (Producer Ms)

Order Ms

(Consumer Ms)

Contians 1 Feign Interface

of Inventory Ms

Report Ms

(Cosumer Ms)

Contians 1 Feign Interface

of Inventory Ms

Both can be same

[we can take two different interfaces having the same effect]

Summary on Client Type comps (Client comps that are required on MS Intra Communication)

note:: Feign interface can take 1 or more methods.. if consumer MS wants to consume only one rest operation of the Producer MS then we take feign interface having one method.. if the Consumer MS wants to consume "n" rest operations of Producer MS then we need to place "n" methods in Feign Interface.

it

support for

Client Type comp

Required

Is Abstract Client

industry

Load Balancing

Annotation

or Concrete Client

standard or not

DiscoveryClient

no

@EnableDiscoveryClient concrete client

not

(main class)

required dependnecy (iar files) Eureka Discovery Cleint starter

operations

Load Balancer Client (LBC)

yes

@EnableDiscoveryClient concrete client (main class)

not

FeignCleint (best)

yes

@Enable DiscoveryCleht, abstract client

Open Feign starter

@EnableFeighClents

yes

(Intenrally InMemory

(main class

class proxy will be generated)

Eureka Discovery Client

starter

=>calfetinstances (to get

Service Instance and

use RestTemplate to make http calls

=>same as above but method

is choose(-) for less load factor instance

=>The Inmemory Proxy class for @FeignClient interface will take

@FeignClient(---)

Eureka DiscoveryClient starter

getting service Instance and making http calls

(interface level)

Why we are adding DiscoveryClient Starter along with Open Feign starter while working with Feign Clients?

an

(jar files)

Ans) Feign Client is abstract client whose logic will be generated in the InMemory Proxy class that implements the given interface.. In that logic it internally1£B (LoadBalanceClient) support

to get LessLoadfactor instance from EurekaServer.. So We need to add Eureka DiscoveryClient starter to the Project. to interact with EurekaServer using the Load BalancerClient code

Every client comp code is supporting code for Consumer MS, So we must register Consumer MS to the Eureka Server Can we take multiple feign interfaces with same Serivce Id?

Ans) yes we can take... but it is unneccessary becoz u r duplicating the work

Q) What is diff between @EnableEurekaClient and @EnableDiscoveryClient?

Ans) @EnableEurekaClient is not avaiable"spring cloud api from spring boot 3.x

as alternate they have given @EnableDiscoveryClient from spring boot 3.x

Q) In Client Comps of Microservices what is the difference b/w Concrete client and abstract client?

=>@EnableEurekaClient upto spring boot 2.x =>@EnableDiscoveryClient

spring boot 3.x

Ans) if the Programmer is writing all the logics(serachingtarget MS, gettting service Instance of target MS, getting the details, generating the http calls to interact with target MS) manually by taking seperate class as spring bean then it is called Concrete Client type Comp

eg:: Discovery Client and Load BalanceClient

=>if Programmer is just providing basic details in the form interface and and its method declarations and all the logics of searching and getting target MS and also commiunicating with Target MS are generated dynamically in the InMemory Proxy class ..then that Client type is called abstract client. eg:: FeignClient.

as

Spring Cloud Config Server

========:

======

=> It is also called Configuration server(CS) and this is useful to get common key-value pairs required for the multiple micro services from the common place.. i.e instead of placing same key-value pairs in multiple micro services.. we can place them in common place and we can get them through configuration server for multiple micro services

=> These common key-value pairs are generally DB conneection properties, email properties, security properties and etc.. which are required as same properties in mutltiple

in

MicroServices..

=> We place these common key-value pairs seperate properties file outside of all MicroServices Projects and

we take seperate project for Configuration Server having one application.properties and this file will be linked with that common/seperate properties file.

spring boot

note:: In MicroServices architecture, the microServices can not be SOAP based web services, they are always Restful services

=> WE need to create Configuration server Project (Also Project) adding "ConfigurationServer" dependendency and this Configuration server by default runs on the Port number 8080 and the recomanded port number: 8888

=> We can make Configuration server Project getting common key-value pairs in two ways

External

#MS1

(CS sever)

config

ConfigServer

client dependency

#MS2

(GitLab, GitHub, BitBucket,...) application.properties (External common properties file)

(best) a) Using External

config

pne we can place common properties file in GIT accounts like github, gitlab(best), bitbucket,..) (Good for all env.. dev, test, uat, prod)

chent dependency

#MS3

confgiserver dependency

b) Using Native

i.e we can place common properties file in Local File System Drives like E:,D: drives and etc.. (Generally used in dev env.. bit)

config

as

=> The real micro service projects that want to use Configuration server managed common properties file content (key=value pairs) must be added with "Configuration Client " dependency.

MicroService#1

applicajten.properties (Local)

key1=val1

(?a2) (not there)

#4

(starter).commom properties

@Value("${key}"(a1) private String data (9) ConfigClient Dependency (starter) MicroService#2

application.properites (Locan key2=val2

(a3) @Value("${key}") (a) private String data; ConfigClient Dependency

#4

Eureka Server Project #2

(all MicroServices

will be published to

This Eureaka Server)

Spring Cloud Configuraiton Server Project

#3

client dependency

(or)

Native Config

fetmstapplication.properties)

(native common properties file)

=>Add Configuration Server dependency to Configuration Server spring

boot project who actually connects and reads the values from the common properties file placed in Code Repository accounts or in FileSystem

=> Add Configuration Client dependency to McirServices Projects who uses the common properties through Configuration server.

(a5)

(a7)

External Config file

(GITHUB, GITLab and etc.. Accounts) #1 application.properties key=val (a6?)

(or)

(these are

note:: The application.properties file that is having common key =value pairs in the Code Repositories like GITLAB/GITHUB and etc.. is called external config file

common key=val note:: The application.properties file that is having common key =value pairs for all pairs in the config server project file system is called native config file

Ms Projects)

MicroService#3

application.properties

(Local)

@Value("${key}")

key3-val3

#4

private String data; ConfigClient Dependency

application.properties (a4) maintains link with external/native

Config Server Dependency

(starter)

(Spring Cloud Configuration server

will not be published Eureka server]

config file

#1 to #4 :: Order of development for the Apps/Projects a1 to a9 :: Flow of the execution

Native Config file (Local File System drives) E:\config\application.properties

the application.properties of external config /native config can contain the following properties like mail properties, jdbc properties, jpa properties, data source properties, security properties and etc..

key=val

#1

(these are

common key=val pairs for all

Ms Projects)

Example Application on using GitLab Account maintainfExternal Config file through spring colud Config server

step1) create application.proeprties (External config file) in Git Lab account

-> go gitlab.com

-> register/singup account, verify through email address

-> singin/Login to git lab account by submitting username, password

that were created above

-> create new Project giving Project name

Menu bar ---> Projects ---> create new Project ----> Blank Project-->

project name :: CsProj1 --->select public ---> create project.

->add applicaiton.properties file in that Project

Go to home page CsProj1 ---> + ---> new file ---> application.properties

add --> key-value pairs

application.properties

dbuser=system

dbpwd=manager

commit.. chanages

any thing can be taken

as the key-value pairs

-> Gather GitLab account Project url (http$url)

Go to CsProj1 home page ---> clone ---> gather url ::

https://gitlab.com/nataraz/csproj1.git

protocol domain

note:: if we create private Project,then we need to pass git lab username, password

in the configserver project otherwise not required

gitlab

project name

username

SpringBootMsProj05-EurekaServer [boot]

> Spring Elements

> JAX-WS Web Services

#src/main/java

com.nt

>ServletInitializer.java

>SpringBootMsProj05Eureka ServerApplication.java

#src/main/resources

static

templates

application.properties

>src/test/java

> JRE System Library [JavaSE-11]

>

Maven Dependencies

Deployment Descriptor: SpringBootMsProj05-EurekaServer

> Deployed Resources

step2) create Eureka server Project in Eclipse IDE

(add :: Eureka server as dependency)

--> place @EnableEurekaServer on the top of main class

--> add the following entries in application.properties

 # server port

server.port=8761

#disable registration and fetching eureka.client.register-with-eureka=false eureka.client.fetch-registry=false

step3) create Spring Cloud Configuration server Project (ConfingSeever Project)

(add Config server Dependency)

(select from spring cloud config section)

->add @EnableConfigServer on main class

-> add the following entries in application.properties file

Available:

config ser

Spring Cloud Config

Config Client

✓Config Server

server port

>src

> target

w HELP.md

mvnw

mvnw.cmd

✓MS SpringBootMsProj05-ConfigServer [boot]

> Spring Elements

Selected:

Web Services

/java

X Config Server

it

vletInitializer.java

ringBootMsProj05ConfigServerApplication.java

server.port=8888

Provide Link to GitLib user account

spring.cloud.config.server.git.uri=https://gitlab.com/natarаz/csproj1.git

Link Url

step4) create Mutliple MicroService Projects having the following dependencies

Eureka

a) spring web b) Discovery Client, c) Config Client (new)

(select from spring Cloud Config section)

Ms#1

-> add @Enable EurekaClient on the main class

#Ms2

-> add the following entries in application.properites file

server port (MS Port)

server.port=9900

service name or applicaiton name

spring.application.name=EMP-SERVICE

#provide Eureka server Url to register Eureka server

/resources

>

templates

src/test/java

application.properties

> JRE System Library [JavaSE-11]

> Maven Dependencies

Deployment Descriptor: SpringBootMsProj05-ConfigServer

>

>

Deployed Resources

> src

> target

WHELP.md mvnw mvnw.cmd Mpom.xml

server using the same process.

We can use GITHUB, BitBucket also

as

External Configuration for Configuration

note:: while working with GIT hub

there is no need of passing.git

in the link url of application.properties

note: if the GIT or BitBucket Project is the private project where this is placed external config file (application.properties) of ConfigServer concept then we need to submit the username and password details

eureka.client.service-url.default-zone=http://localhost:8761/eureka

JIe

 # To make Ms Connecting to 8888 port number ConfigurationServer (required from spring boot 2.4 onwords)
spring.config.import-optional:configserver:

->Develop one RestController conusuming the vlaues of extenal config file (GibLab account application.properties file) //controller class

package com.nt.controller;

import org.springframework.beans.factory.annotation.Value; import org.springframework.web.bind.annotation.GetMapping; import org.springframework.web.bind.annotation.RequestMapping; import org.springframework.web.bind.annotation.RestController;

@RestController

@RequestMapping("/emp")

public class EmployeeOperationsController {

SpringBootMsProj05-EmpRestService [boot]

> Deployment Descriptor: SpringBootMsProj05-EmpRestService

>Spring Elements

>JAX-WS Web Services

src/main/java

✓com.nt

>ServletInitializer.java

> SpringBootMsProj05EmpRestServiceApplication.java

com.nt.controller

> EmployeeOperationsController.java

#src/main/resources

static

application.properties

templates

>src/test/java

> JRE System Library [JavaSE-11]

>

Maven Dependencies

> C Deployed Resources

@Value("${dbuser}")

> src

private String user;

> target

key placed in gitlab account

WHELP.md

(Extenal Config server)

mvnw

mvnw.cmd

M pom.xml

@Value("${dbpwd}")

private String pass;

@GetMapping("/show")

return "Data Collected throgh Config Server ::"+user+"----"+pass;

public String showData() {

}

}

application.properties

 # server port (MS Port)

server.port=9901

 # service name or applicaiton name

spring.application.name=CUST-SERVICE

#provide Eureka server Url to register Eureka server

eureka.client.service-url.default-zone=http://localhost:8761/eureka

 # To make Ms Connecting to 8888 port number ConfigurationServer (required from spring boot 2.4 onwords)

spring.config.import=optional:configserver:

-> add @EnableEurekaClient on the main class

//CustomerOperationsController.java

package com.nt.controller;

import org.springframework.beans.factory.annotation.Value; import org.springframework.web.bind.annotation.GetMapping; import org.springframework.web.bind.annotation.RequestMapping; import org.springframework.web.bind.annotation.RestController;

SpringBootMsProj05-CustomerRestService [boot]

> L. Deployment Descriptor: SpringBootMsProj05-Customer RestService

>

Spring Elements

>JAX-WS Web Services

#src/main/java

spring.cloud.config.server.git.uri=https://gitlab.com/nareshit_ameerpet/csproj03.git

spring.cloud.config.server.git.username=gituser

spring.cloud.config.server.git.password=gitpassword

com.nt

>ServletInitializer.java

> SpringBootMsProj05EmpRestServiceApplication.java

com.nt.controller

> CustomerOperationsController.java

#src/main/resources

=>To link ConfigClient (Ms) with Config server before spring boot 2.4 we need to place the following entries in the application.properties file

spring.cloud.config.enabled=true

spring.cloud.config.uri=http://localhost:8888

=> From spring boot 2.4 and if the config server is running on the port number 8888

spring.config.import=optional:configserver:

=> From spring boot 2.4 and if the config server is running on other than 8888 port number

static

templates

application.properties

@RestController

>

src/test/java

@RequestMapping("/cust")

> JRE System Library [JavaSE-11]

>

Maven Dependencies

public class CustomerOperationsController {

>

Deployed Resources

> src

@Value("${dbuser}")

> target

WHELP.md

private String user;

mvnw

@Value("${dbpwd}")

mvnw.cmd

M pom.xml

private String pass;

@GetMapping("/display")

}

public String displayData() {

return "(Customer)Data Collected throgh Config Server ::"+user+"----"+pass;

}

step5) Run the Applications in the following order

->Run Eureka server Project

-> Run All Ms Projects

-> Run Config Server Project

-> Go to EurekaSever Home page and modify the URL both

Emp, Cust Ms services

http://desktop-iudaavl:9901/cust/display

http://desktop-iudaavl:9900/emp/show

---> To generate request Cust M..

---> To generate request Employee Ms

spring.config.import-optional:configserver:

spring.cloud.config.uri=http://localhost:8899

Making Multiple MicroServices getting common data from Native Config file (Local system drives)

With the support Spring Cloud Configuration server

=> It is suitable only in Dev, Test env.. but not in UAT, production env..

=>Use this in dev, test env.. if ur not ready with GIT Accounts

=> Generally we place this Native Configuration related application.properties file

in the spring Cloud Configuration project itself (which also uses Local system Drives)

Example App

==========

step1) Develop Eureka Server App

(same as previous App)

step2) Develop Configuration Server

(add Config server Dependency)

(select from spring cloud config section)

->add @EnableConfigServer on main class

=>In all environments, the industry prefers using

External config server file by placing in GIT accounts

-> add new applicaiton.properties by creating a folder like "config" in_src/main/resources folder

to keep common key-value pairs required for all Ms projects.

src/main/resources

✓ config

application.properties

dbuser=system

dbpwd=manager

src/main/java, src/main/resources folders

of maven project will be placed in classpath by default.

->add the following entries in application.properties file of src/main/resources folder

#src/main/resources

✓ config

application.properties

static

templates

(this file name is fixed

but the location is ur choice)

application.properties (This file name and location both are fixed)

 # server port

(this is regular file)

server.port=8888

#active native profile of its Parent Project

spring.profiles.active=native

 # specify "config" folder of classpath as NativeConfig Location for Configuration Server

spring.cloud.config.server.native.search-locations-classpath:/config

Native Cofig Location

The Parent project of Configuration server is having profiles.. the default

profile is designed to get Linked with Git Accounts i.e taking Extenal Configuration where as "native" profile is designed to get linked with Native Config (local drives)

So we are activating native profile.

step3) Develop Multiple MicroServices

(same as previous)

step4) Execute the Applications/Projects in the following order.

->Run Eureka server Project

-> Run Config Server Project

-> Run All Ms Projects

-> Go to EurekaSever Home page and modify the URL both

Emp, Cust Ms services

http://desktop-iudaavl:9901/cust/display

---> To generate request Cust Ms http://desktop-iudaavl:9900/emp/show ---> To generate request Employee Ms