Working with RefreshScope using @RefreshScope Annotation

In

(or) Local Config file

=>The modifications GitLab/GITHUB Extenal Config file content will reflect to all the MicroServcies olnly when festart the Config serrver and all Mses But Restrating Configserver every time for each modification and all Ms

done in External Config is not a recomanded process.

file

-

(note:: Same problem is there w.r.t Native Config file)

=>To overcome that problem we can use RefreshScope (@RefreshScope) with support of spring boot Actuators (readymade endpoints/ready made Microservices given by spring boot)

with

Procedure to work @RefreshScope

====

add

step1) spring boot actuator dependency to our MicorServices Projects

<dependency>

(Config Client)

<groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

step2) Activate the readymade actuator "refresh" in our MS Project (Config Client) adding the entries in application.properties

In

application.properties

# Activate all actuators (*) or only refresh actuator management.endpoints.web.exposure.include=*

step3) place @RefreshScope on the top RestController of In Ms Project

@RestController

@RequestMapping("/emp")

@RefreshScope

public class EmployeeOperationsController {

(refresh actuator)

right click on project ---> properties

---> spring ---> add starter ---> search for actuator and select spring boot actuator.

=>if we create Project in the GIT Lab as the

private Project, we need to pass the GIT Lab account username, password in the application .properties file of ConfigServer Project as shown below

=>Spring boot actuators are given for executing the non-functional features of the Project.. they are readymade microservices .Non functional features are like threaddump info, health info, metrics and etc..

(additional entry)

@RefreshScope

In application.properties of the ConfigServer

spring.cloud.config.server.git.username=<username> spring.cloud.config.server.git.password=<pwd>

Functional features of the Project are like primary logics. eg:: create account, withdraw, deposite and etc..

Non-Functional Features are like Secondary logics eg: Threads Info, Logs Info, memory info, system infra info and etc..

Convenience annotation to put a @Bean definition in refresh scope. Beans annotated this way can be refreshed at runtime and any components that are using them will get a new instance on the next method call, fully initialized and injected with all dependencies.

}

step4) start Eureka Server, Config Server, MsProject in regularfashion

step5) Test the Ms using the regular url

step6)

http://desktop-iudaavl:9910/emp/show

Modify entries in application.properties file of GitLab account (External Config file) open properties file in git lib --> edit web editor ---->

step7) Test the MS using regular url

http://desktop-iudaavl:9910/emp/show

(Modifications does not reflect suprisingly)

step8) Gather EndPoint details of "refresh" spring boot actuator and give POST mode request to it using POSTMAN tool.

"refresh" actuactor url is http://localhost:9910/actuator/refresh

step9)

POST (a)

http://localhost:9910/actuator/refresh (b)

Params

Query Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

KEY

Key

Body Cookies Headers (5)

VALUE

Value

Test Results

(c)

Send

Cookies

DESCRIPTION

Bulk Edit

Description

Pretty

Raw

Preview

Visualize

JSON

(d)

1

2

"dbpwd",

3

"config.client.version"

4

Test the MS using regular url

http://desktop-iudaavl:9910/emp/show

(chanages will reflect)

Reflects by

creting new

object for controller

Ms#1 Project

@RestController

ConfigServer

@RefreshScope

EmpController

(b)

1-1

@Value("${dbpwd}"Y

private String pwd; spring boot

(d)

actuator (refresh)

(H)

(c)

(a)

POSTMAN

Tool FOST http://localhost:9910/actuator/refesh

(e)

http://lolcahost:9910/emp/show

"dbpwd",

(request to MS)

"config.client.version"

=>loin comes here

Git Lab Account

applicaiton.properties

dbpwd=ehee-lion

send

note:: In real projects maintaince, the GITLAB account data common properties will be modified and giving POST mode request to refresh spring boot actuactor will be taken care by devops team

(Project maintainace team)

Q) How to reflect the changes done in external config file (GIT Lab account) (or) the native config file (Local file) to the MS Projects with out restarting the config server and the MS Project?

Ans) we need to use spring' Actuatorefresh" in combination with @RefreshScope annotation...

Spring Boot Actuators

====================

@RefreshScope

(This annotation makes the IOC container to create new object for @RestController of MS for every change that "refresh" actuator gets for Injection)

Production Sever :: The Server in which Application is deployed to give services to endusers

for commercial use is called Production Server. This Server manages the Live Code and Live DB..

Old days :: Dedicated high end System for Producttiion Server recent days :: using Cloud env (rental basis infrastructure) like AWS, Google Cloud and etc..)

Production Enviroment /Setup :: The software and tools that are used in production server

to deploy and run the App is called Production env/setup eg:: jdk s/w, wildfly server/Tomcat server, jenkins, Docker and etc..

note:: The functional features are the primary logics of the App.

eg:: Banking App's opening account, withdraw money, deposite money and etc..

are

The non-fuctional features the secondary logics of the App which additional and optional logics to manage the application effectively

eg:: logging, thread dump info, process dump, heap memory info bean ids info and etc..

EndPoint

: :

The Rest apis/Rest Service /some other service that is ready to use is called Endpoint.. In order take the services from any EndPoint we need gather

Method

End point details like url, method type, path variables and etc.. (Each @RestController is called one EndPoint)

=>Each RestControllere develop spring boot App is called one Endpoint and to use

We

collect

that endpoint we need to the above said details

eg: http://localhost:9910/emp/show (URL), GET (method type), {id}, {name} are path variables, Input type : JSON, output type: String and etc..

Like this we should collect the details

for every repenttion

of the RestController

(These are technically called as endpoint details)

Spring Boot Actuator: It is production ready endpoint (buit-in Rest Service) given by spring boot people to perform non-functional operations on the production env..spring boot project.

eg:: health, info, configprops,beans,logging, refresh and etc.. (generally we get 15+ ready made actuators)

=> Upto spring boot 2.5 we used to see two actuators as activated/enabled actuators by default they are health, info

=> From spring boot_2.6 they are giving only health as defaultly activated actuator.

=> In any Ms Project add "spring boot actuators" dependency in order use spring actuator services.

Spring boot actuators are the production ready microservices to apply the non-functional features of on the production ready spring boot projects

<dependency>

<groupId>org.springframework.boot</groupId> <artifactId>spring-boot-starter-actuator</artifactId>

</dependency>

URL to see all the defaulty activated actuators

http://localhost:9901/actuator

Current App port number

on

Right click spring boot project

--> spring --> add starter -->

select spring boot actuator

You

localhost:9900/actuator

x +

← → C

localhost:9900/actuator

▼ {

▼ "_links": {

▼ "self": {

"href": "http://localhost:9900/actuator",

"templated": false

← → C

localhost:9901/actuator

```
{"_links":{"self": {"href": "http://localhost:9901/actuator","templated": false},
"health":{"href": "http://localhost:9901/actuator/health","templated": false},
"health-path": ["href":
"http://localhost:9901/actuator/health/{*path}","templated":true}}}
```

Example app

=============

=> create Spring Boot Starter Project (As RestService / Not As MicroService)

adding web, actuator dependencies

(No need of adding DiscoryClient, Config Client)

=> Develope RestController as shown below

@RestController

@RequestMapping("/cust")

public class CustomerOperationsController {

},

▼ "health-path": {

"href": "http://localhost:9900/actuator/health/{*path}", "templated": true

},

"health": {

"href": "http://localhost:9900/actuator/health", "templated": false

To get this view

INSTALL

JSON Viewer plugin in

chrome browser

note:: Using Chrome Extension tab also we can install JSON plugin

https://chrome.google.com/webstore/detail/json-formatter-viewer-

and/infnlhnhibphpaljmnnadaldibggkokb

(use this URL to add JSON View plugin to chroime)

}

@GetMapping("/display")

public String displayData() {

}

return "(Customer) Customer Info will be displayed here.. welcome Call Center ";

=> add following entries in application.properties

#server port (MS Port)

server.port=9901

# service name or applicaiton name

spring.application.name=CUST-SERVICE

=> Run the App as spring boot App

=> get the defaultly activated list of actuactors.

note:: spring boot actuators are readymade

rest services given by spring cloud holding

infomation about current projects and its executing environment.

http://locahost:9901/actuator

To get this content as the formatted content

☆

*

install JSON Plugin to Chrome browser software.

localhost:9901/actuator

```
{"_links":{"self": {"href": "http://localhost:9901/actuator","templated": false},
"health":{"href": "http://localhost:9901/actuator/health","templated": false},
"health-path' ["href":
"http://localhost:9901/actuator/health/{*path}","templated":true}}}
```

health

=====

=> This actuator gives health metrics info like Currenet App is up or down?

=>Using this actuator,we can also get more detais related memory like Diskspace, free space used space and etc..

=> It is default activated spring boot actuator.

=>These actuators services are very useful for Project maintainace team generally that is DevOps Team

```
{"status":"UP"}
```

localhost:9901/actuator/health

=>To get memory details about the application we need to add one more supporting entries in application.properties.

In application.properties

note: Before arrival of spring boot to market, we use d develop these kind of non-functional features/services using aop module manually.. Since spring boot is giving actuators as the readymade microservices representing various non-functional features we can say there is no need of learning aop module in the current situation

note:: Before the arrival of Spring boot actuators we used work with AOP module and lots of third party tools to arrange non-functional features for DevOps team (Project maintenance team). After the arrival of Spring boot actuators no such requirement is there (Currently spring AOP /spring boot AOP module lots its significance in the market)

#To get Memory details default using health actuator management.endpoint.health.show-details-always

(the possible values are always,never(default), when-authorized)

← C

localhost:9901/actuator/health

(best)

☆

```
{"status":"UP", "components": {"diskSpace":{"status": "UP", "details":{"total":
185532936192, "free": 145165168640,
"threshold":10485760,"exists":true}},"ping":{"status":"UP"}}}
```

management.endpoint.health.show-details-always

Logged in

(always gives health metrics(Memory details) though u have not

to the App)

management.endpoint.health.show-details- never

(Does not give health metrics)

management.endpoint.health.show-details-when-authroized

(gives health metrics(Memory details) only for the logged in user of the App i.e authentication

and authorizations are completed.)

To disable any atcuator

In application.properties

management.endpoint.<endpoint-id>.enabled=false

(default is true)

these are like health, beans,info and etc..

To enable certain actutor which is not activated by default

In application.properties

management.endpoints.web.exposure.include-info

(Activates only info actuator)

management.endpoints.web.exposure.include=info,health

(Activates only info,health actuators)

management.endpoints.web.exposure.include=* (Activates all the actuators)

when all actuators are activated the list looks like this

note:: Spring boot actuators can be applied on Restful services or on MicroServices

management.endpoint.<endpoint-id>.enable is deprecated in latest versions of the spring boot.. as alternate use management.endpoint.<endpoint-id>.accesss=none

The possible values are :: none,unrestricted, readOnly

← →

localhost:9901/actuator/

```
{"_links":{"self": {"href": "http://localhost:9901/actuator","templated": false},
"beans ":{"href": "http://localhost:9901/actuator/beans","templated":false},"caches":
{"href": "http://localhost:9901/actuator/caches","templated": false},
"caches-cache":{"href":
"http://localhost:9901/actuator/caches/{cache}","templated":true}, "health": {"href":
"http://localhost:9901/actuator/health","templated": false}, "health-path":{"href":
"http://localhost:9901/actuator/health/{*path}","templated" : true}, "info": {"href":
"http://localhost:9901/actuator/info","templated":false},"conditions":{"href":
"http://localhost:9901/actuator/conditions", "templated": false},
"configprops-prefix": {"href":
"http://localhost:9901/actuator/configprops/{prefix}","templated":true},"configprops":
{"href": "http://localhost:9901/actuator/configprops","templated": false}, "env-
toMatch":{"href": "http://localhost:9901/actuator/env/{toMatch}","templated" : true},
"env":{"href": "http://localhost:9901/actuator/env","templated":false},"loggers-name":
{"href":"http://localhost:9901/actuator/loggers/{name}","templated":true},"loggers":{"
href": "http://localhost:9901/actuator/loggers","templated": false}, "heapdump":
{"href": "http://localhost:9901/actuator/heapdump","templated": false},
"threaddump":{"href":
"http://localhost:9901/actuator/threaddump","templated":false},"metrics- {"href":
"http://localhost:9901/actuator/metrics","templated": false},
"scheduledtasks":{"href":
"http://localhost:9901/actuator/scheduledtasks","templated":false},"mappings":
{"href":"http://localhost:9901/actuator/mappings","templated": false}}}'
```

```
requiredMetricName":{"href":
"http://localhost:9901/actuator/metrics/{requiredMetricName}","templated":true},"metri
cs":
```

from spring boot 3.x, we are getting

an additional actuator called sbom

Info

of

=>Gives info about current application in the form JSON content when use request to this "info" atuator
=>For that we need to maintain information about application in application.proeprties file having fixed prefixes

in keys (thatis info.app)

example

=> keep spring rest app ready (same as previous)

=> activate "info" or all actuators

In application.properites

management.endpoints.web.exposure.include=info

(or)

management.endpoints.web.exposure.include=*

=> add more info about the application.in application.properties having fixed prefix in keys (info.app)

in application.properties

can be any thing

# info about application

info.app.name=CUST-Service

prefix

info.app.id=45467

(fixed)

info.app.size=10modules

note:: if the project maintainance team

info.app.vendor=Naresh IT

info.app.createdBy=Team-S

=> enable info enviorment to read about current app

In application.properties

management.info.env.enabled=true

=>Test the info acuator Application

← C

localhost:9901/actuator/info

which is DevOps team wants to know more info about current Project with out having access to source code.. then they get that info through "info" actuator.

(very very useful to know which build version code

is under execution)

```
{"app":{"name":"CUST-Service", "id": "45467","size":"10modules", "vendor":"Naresh IT",
"createdBy":"Team-S"}}
```

To disable info actuator

management.endpoint.info.enabled=false

(or)

management.endpoint.info.access=read-only (or) none

shutdown actuator

=======================

=> This actuator does not come in the list of actuators though we use "*" for

management.endpoint.web.exposure=* in application.proepeties.. we need enable it seperately

management.endpoints.web.exposure.include=*

management.endpoint.shutdown.access=UNRESTRICTED

test1:: http://localhost:8093/BillingMs/actuator/

=>gives "shutdown" in the list of actuators

=>Shutdown actuator is really useful to shutdown the project gracefully from remote location (useful for DevOps team)

=> For this we need to give POST mode request to "Shut Down" Actuator

(a)

POST

http://localhost:8093/Billing Ms/actuator/shutdown (b)

Send

(c)

Params

Authorization

Headers (8) Body

Scripts

Settings

Query Params

Key

Value

Body Cookies Headers (5)

{} JSON ✓

▷ Preview

Test Results

Visualize

```
v{
"message": "Shutting down, bye..."
}
```

200 C

By adding this actuator to the MS Project, we can shutdown the MS with out touching the MicroService