Activating spring boot profiles using the "System" property "spring.profiles.active"

=> we can use -D to pass user-defined System properties in the execution of the application.

=> In eclipse IDE, use "Program Arguments" section pass the command line args

=> In eclipse IDE, use "VM Arguments" section pass the system properties

note: Should be given in eclipse IDE

as VM arguments using Run Configurations.

example App

==========

step1) keep the Spring Profiles App ready

step2) comment spring.profiles.active key in the application.properties

step3) Run the main class using VM arguments option as shown below

Right click on main class ---->run ---> run as --->run configurations ----> arguments tag ---->

*** we can see these sections in run as --> run configuration option...**

VM arguments:

-Dspring.profiles.active=dev

---> apply ----> run

Variables...

Q) if we activate two different profiles using application.properties and using system property then which will be taken as the final profile?

Ans) The "profile specified" in the System property will be taken as the final value.

Creating profiles with support of multiple YML files

http://mageddo.com/tools/yaml-converter

✓ BootProj11-LayeredApp-Profiles-MultipleYMLFiles [boot]

>Spring Elements

#src/main/java

com.nt

BootProj03LayeredAppRealtimeDiApplication.java

com.nt.controller

> PayrollOperationsController.java

com.nt.dao

> EmployeeDAO.java

>MySQLEmployeeDAOImpl.java

> OracleEmployeeDAOImpl.java

com.nt.model

> Employee.java

>

>

com.nt.service

EmployeeMgmtServiceImpl.java

EmployeeMgmtService.java

#src/main/resources

application.yml

application-dev.yml

application-prod.yml

application-test.yml

application-uat.yml

#src/test/java

>JRE System Library [JavaSE-16]

Maven Dependencies

> src

> target

WHELP.md

mvnw mvnw.cmd Mpom.xml

application-dev. yml

#jdbc properties

spring:

datasource:

driver-class-name: com.mysql.cj.jdbc.Driver

url: jdbc:mysql:///NTSPBMS616DB

username: root

password: root

dbcp2:

max-total: 100

initial-size: 10

max-conn-lifetime-millis: 100000

type: org.apache.commons.dbcp2.BasicDataSource

application-test.yml

spring:

datasource:

driver-class-name: com.mysql.cj.jdbc.Driver

username: root

url: jdbc:mysql:///NTSPBMS616DB

type: com.mchange.v2.c3p0.Combo Pooled DataSource

password: root

c3P0:

maxSize: 100

minSize: 10

application-uat.yml

1 spring:

2 datasource:

driver-class-name: oracle.jdbc.driver.Oracle Driver

application-prod.yml

1 spring:

2 datasource:

driver-class-name: oracle.jdbc.driver.Oracle Driver

keepalive-time: 100000

maximum-pool-size: 100

3

4

username: system

3

5

url: jdbc:oracle:thin:@localhost:1521:xe

4

hikari:

6

oracleucp:

5

7

6

8

time-to-live-connection-timeout: 100000

7

9

8

10

type: oracle.ucp.jdbc.PoolDataSourceImpl

9

11

10

12

max-pool-size: 100

min-pool-size: 10

password: tiger

application.yml

1 #Activate the profile

2 spring:

3 profiles:

minimum-idle: 10

username: system

url: jdbc:oracle:thin:@localhost:1521:xe

password: tiger

4

active: prod

Working with single yml file having multiple profiles

==========

file

=> we can place multiple profiles in a yml by taking --- as a seperator in the file

BootProj11-LayeredApp-Profiles--single-yml-file [boot] [NTSPBMS715 master]

> Spring Elements

src/main/java

com.nt

BootProj03LayeredAppApplication.java

com.nt.controller

> PayrollOperationsController.java

com.nt.dao

>IEmployeeDAO.java

> MySQLEmployeeDAOImpl.java

>OracleEmployeeDAOImpl.java

com.nt.model

> Employee.java

com.nt.service

>EmployeeMgmtService.java

> EmployeeMgmtService.java

src/main/resources

application.yml

> src/test/java

>JRE System Library [JavaSE-17]

> Maven Dependencies

> src

>

target

w HELP.md

mvnw

mvnw.cmd

pom.xml

application.yml

#Activate the profile spring:

profiles:

active: test

To create each profileml file use

=>spring:profiles key in old versions (spring boot 1.x and spring boot 2.x) =>spring:config:activate:on-profile (In spring boot 3.x)

spring:

config:

activate:

on-profile: dev

datasource:

driver-class-name: com.mysql.cj.jdbc.Driver

url: jdbc:mysql:///NTSPBMS616DB

username: root

password: root

dbcp2:

max-total: 100

initial-size: 10

max-conn-lifetime-millis: 100000

type: org.apache.commons.dbcp2.BasicDataSource

spring:

config:

activate:

on-profile: test

datasource:

driver-class-name: com.mysql.cj.jdbc.Driver

username: root

url: jdbc:mysql:///NTSPBMS616DB

type: com.mchange.v2.c3p0.Combo Pooled DataSource

password: root

c3P0:

maxSize: 100

minSize: 10

spring:

config:

activate:

on-profile: uat

datasource:

driver-class-name: oracle.jdbc.driver.Oracle Driver

username: system

url: jdbc:oracle:thin:@localhost:1521:xe

oracleucp:

max-pool-size: 100

time-to-live-connection-timeout: 100000

min-pool-size: 10

type: oracle.ucp.jdbc.PoolDataSourceImpl password: tiger

spring: config:

How many ways are there to activate profiles in spring boot app?

Ans) There are multiple ways to use .. but mainly 3 approaches can be considered

a) using application.properties/yml (Best)

b) using System property (-Dspring.profiles.active=dev)

c) Using Programatic approach (by using the methods of SpringApplication class)

b) using System property (-Dspring.profiles.active=dev)

Not at all good approaches

if the project is in form of jar file

Right click in main class---> run as----> run configuratiions ---> arguments tab VM arguments: (system properties)

-Dspring.profiles.active=prod

cmd> java -jar -Dspring.profiles.active=dev App1.jar

activate:

on-profile: prod

datasource:

driver-class-name: oracle.jdbc.driver.Oracle Driver

hikari:

keepalive-time: 100000

maximum-pool-size: 100

minimum-idle: 10

username: system

url: jdbc:oracle:thin:@localhost:1521:xe password: tiger

--> apply ---> run

note:: Active profiles info will be stored in Enviromement object which is built-in object of IOC container...

In main class

Active profiles

info will be

stored in

Environment obj

Environment env=ctx.getEnvironment();

irrepsective of approach

System.out.println("active profile name::"+Arrays.toString(env.getActive Profiles()));

we are using to activate the

profile.

c) Using Programatic approach (by using the methods of SpringApplication class)

in main method of main class

//create object for SpringApplication class

=>Environment object is the IOC container managed special object

cum spring bean which holds the following details

a) Info gathered from the properties profiles/yml files

b) Info about System properties

c) Info about Enviroment variables

d) Current active profile information

SpringApplication application=new SpringApplication (BootProj03 LayeredAppApplication.class);

application.setAdditionalProfiles("uat");

//get IOC container

//specifying the activte profile

ApplicationContext ctx-application.run(args);

... other lines

we

if activate 3 different profiles by using 3 different approaches can u tell me which approach specified profile will be activated?

Ans) It will activate multiple beans by collecting from multiple profiles that are specified in different approaches activated profiles.. so mismatching of profiles takes place and

possibility of getting ambiguity problem is there... So use single approach at a time to activate the profile

we

if targt spring bean is having multiple possible dependent spring beans then how can specofy specific dependent spring bean with out touching the java source of the project/application? <alias>.tag (xml file)

a) using spring bean alias concept + spring bean cfg file + properites file/yml file

(In spring boot, 100% code configuration approach also we need to use xml file support)

b) using spring or spring boot profiles (Best)

of

=>Generally we do not run the same copy Project

in different profiles simultaenously.. But we can run

different copies same project on different profiles

at a time by taking those copies of the Project in different machines

(bad approach)

if there is no matching properites file/yml file for the specified active profile then it takes application.properites file as default fallback Child Profiles or Profiles Include

==========

=>while taking application.properties file as fallback properties file having fallaback profile inforamtation.. instead of writing repeated key and values in application.properties which

profile file

are already there in another profile specific properties file.. we can include that profile name applicable even using

in application.properties file as shown below.

this concept is yml

application.properties

==========

#activate spring profile

spring.profiles.active=staging

here application.properites

default fallaback profile

becomes child profile to

the specified parent profile "prod"

 # include mutliple keys from other profile specific proerties fiels

spring.profiles.include-prod

(This inclusion makes application.properties related

fallback profile as child profile for prod profile.)

=>we do not have application-staging.properties file

so it will take application.properties as the

fallback file.. but In that file we are no not and values directly rather we getting by including another profile (prod)

application-dev.properties

application-test.properties

application-uat.properties

application-prod.properties

Q) What is the meaning of @Profile("default") or what is default profile?

same as previous Application..

note:: we can include other profiles info only in application.properties/yml

=> if no active profile is specified then all the beans of "default" profile will be instantiated and used

=>It is recomanded to have complete setup of all spring beans to execute in "default" profile ..i.e when no active profile is specified or no child profile is specified

=>we can make certain spring bean working for specific profiles and also for default Profile

as shown below..

i.e not possible in profile specific properties files/yml files (like application-uat.properites/yml,

=>if no active profile is specified then the spring

beans not having @Profile and having @Profile("default") will be instantiated automatucally,, and info will be collected only from application.properties/yml file

application-prod.properties/yml,....)

@Repository("empDAO1")

@Profile({"dev","test","default"})

To make app running with out

active profiles.. we just need to comment

public class MySQLEmployeeDAOImpl implements IEmployeeDAO {

}

.is

#spring.profiles.active=dev of applicaiton.properites file "default" is not profile name.. it the env.. or setup of spring /spring boot that executes when no active profiles are specified.

What is the difference b/w fallback profile and default profile and also child profile?

Ans) Fallback Profile :: The profile that executes when requested active profile is not avaiable (if spring.profile.active key specified with wrong/ unavaiable profile name) default profile :: The profile that executes when there no activate profile in the application. (if spring.profile.active key is commented) child profile/profile include :: The Profile that is included in application.properties by specifying the profile name to reuse keys and values specific profile in application.properties indirectly as fallback profile.

Can we keep multiple spring beans of same category/type(having common super class/implementing interface) in default profile?

Ans) No, Not possible.. Conflicts /Ambiguity Problems may come

=>Spring /spring Boot gives

=>if spring.profile.include is used

in application.properties/yml file

org.springframework.beans.factory.NoUniqueBeanDefinitionException: expected single matching bean but found 2: cds,tcp

will come more Dependent of same type found to inject to target bean

=> spring /spring boot gives "org.sf.beans.factory.NoSuchBeanException" if the given bean id spring bean is not avaiable.

Q) what is difference b/w not adding @Profile for spring bean and adding @Profile("default") for spring bean?

is

Ans) if @Profile is not added for spring bean that will work for all. profiles when profile activated.. it will even work activated and also for child profile that is activated eg: service, controller classes

if no profile is

=> @Profile("default") based spring bean will work only when active profiles are not specified. (if spring.profiles.include, spring.profiles.active keys in application.properties file is missed) =>In real projects we keep DS classes, DAO classes in specific profile or in default profile becoz persistence logic changes DB s/w to DB s/w becoz SQL queries are DB s/w dependent Queries => In real projects, we do not keep controller, service classes in any profile becoz they common for all profiles and not dependent to any Db s/w.

are

Conclusion on the Profiles