Date : 21/09/2020

Spring Boot 7AM

Mr. RAGHU

------------------------------------------------

**Profiles using YAML File**

i. Multi YAML file

application-[ProfileName].yml

ii.\*\*\* Single YAML File

application.yml

k:v

---

spring:

profiles: QA

k:v

---

spring:

profiles: PROD

k:v

---

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

Profiles can also be implemented using YAML file. It povides support like Properties file

ie creating one YAML files for one profile, using syntax:

application-[profileName].properties. [If no profile name is provided then it is default]

--Ex#1 Using Multiple YAML Files -------

#1. Create one Starter project

Name : SpringBoot2ProfilesYamlExOne

Dep : Lombok

#2. Rename application.properties file

> Rightclick on application.properties > Refactor > Rename > Enter new name

application.yml > Finish

> Right click on src/main/resource > new > File > Enter name : application-qa.yml > Finish

--application.yml--

my:

app:

driver: ORACLE-DEV

url: SAMPLE-DEV

--application-qa.yml--

my:

app:

driver: ORACLE-QA

url: SAMPLE-QA

#3. Runner class

package in.nareshit.raghu.runner;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.stereotype.Component;

import lombok.Data;

@Component

@Data

@ConfigurationProperties(prefix = "my.app")

public class DataReadRunner implements CommandLineRunner {

private String driver;

private String url;

@Override

public void run(String... args) throws Exception {

System.out.println(this);

}

}

#4. RunConfigurations : --spring.profiles.active=qa

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

**Profiles using one YAML File**

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

We can hanlde multiple profiles data(k:v) using one YAML file with symbol 3 dash(---).

If we write 3 dash symbols it indicates profile is changed (new profile started).

We should also provide profile name using

spring:

profiles: <name>

=====application.yml=======

# This is DEFAULT Profile

my:

app:

driver: ORACLE-DEV-NEW

url: SAMPLE-DEV-NEW

---

# This is QA Profile

spring:

profiles: qa

my:

app:

driver: ORACLE-QA-NEW

url: SAMPLE-QA-NEW

---

# This is Production Profile

spring:

profiles: prod

my:

app:

driver: ORACLE-PROD-NEW

url: SAMPLE-PROD-NEW

===Runner class=======

package in.nareshit.raghu.runner;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.stereotype.Component;

import lombok.Data;

@Component

@Data

@ConfigurationProperties(prefix = "my.app")

public class DataReadRunner implements CommandLineRunner {

private String driver;

private String url;

@Override

public void run(String... args) throws Exception {

System.out.println(this);

}

}

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

\*) Even YAML file can also be used/read with @Profile annotation.

\*) Note:

=> @Component // creating object

=> @Profile("qa") // provide profile name

=> @ConfigurationProperties(prefix = "my.app") //read key:vals

=> @Data // set/get, toString, equals..

--code--

1. application.yml

# This is DEFAULT Profile

my:

app:

driver: ORACLE-DEV-NEW

url: SAMPLE-DEV-NEW

---

# This is QA Profile

spring:

profiles: qa

my:

app:

driver: ORACLE-QA-NEW

url: SAMPLE-QA-NEW

---

# This is Production Profile

spring:

profiles: prod

my:

app:

driver: ORACLE-PROD-NEW

url: SAMPLE-PROD-NEW

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

2. Service Interface

package in.nareshit.raghu.service;

public interface ExportService {

public void export();

}

3. ServiceImpls

package in.nareshit.raghu.service;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.context.annotation.Profile;

import org.springframework.stereotype.Component;

import lombok.Data;

@Component // creating object

@Profile("qa") // provide profile name

@ConfigurationProperties(prefix = "my.app") //read key:vals

@Data // set/get, toString, equals..

public class ExcelExportService implements ExportService {

private String driver;

private String url;

@Override

public void export() {

System.out.println("FROM EXCEL");

System.out.println(this);

}

}

----------

package in.nareshit.raghu.service;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.context.annotation.Profile;

import org.springframework.stereotype.Component;

import lombok.Data;

@Component

@Profile("prod")

@ConfigurationProperties(prefix = "my.app")

@Data

public class PdfExportService implements ExportService {

private String driver;

private String url;

@Override

public void export() {

System.out.println("FROM PDF");

System.out.println(this);

}

}

-------------

package in.nareshit.raghu.service;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.context.annotation.Profile;

import org.springframework.stereotype.Component;

import lombok.Data;

@Component

@Profile("default")

@ConfigurationProperties(prefix = "my.app")

@Data

public class CsvExportService implements ExportService {

private String driver;

private String url;

@Override

public void export() {

System.out.println("FROM CSV");

System.out.println(this);

}

}

-----------

#3. Runner class

package in.nareshit.raghu.runner;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.service.ExportService;

@Component

public class DataReadRunner implements CommandLineRunner {

@Autowired

private ExportService service;

@Override

public void run(String... args) throws Exception {

service.export();

}

}

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

**StopWatch using Spring Boot**

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

StopWatch : it is used to calculate time taken for a task finish/execution.

API:

StopWatch (C)

org.springframework.util [Package]

Constructor:

StopWatch()

StopWatch(id)

Methods:

start()

start(taskName)

stop()

getTotalTimeMillis()

getTotalTimeSeconds()

prettyPrint()

-------Ex#1----------------------------------------------------

#1. Create new Starter Project

Name : SpringBoot2StopWatchEx

#2. Runner class

package in.nareshit.raghu.runner;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import org.springframework.util.StopWatch;

@Component

public class StopWatchTestRunner implements CommandLineRunner {

@Override

public void run(String... args) throws Exception {

StopWatch watch = new StopWatch();

watch.start();

//watch.start();//invaild --IllegalStateException

for (int i = 0; i < 100000000; i++) {

double result = Math.pow(i+1, i) + Math.pow(3, i) ;

}

watch.stop();

//watch.stop(); //invaild -- IllegalStateException

//skip time calculation for below loop

for (int i = 0; i < 100000000; i++) {

double result = Math.pow(i+1, i) + Math.pow(3, i) ;

}

watch.start();

for (int i = 0; i < 100000000; i++) {

double result = Math.pow(i+1, i) + Math.pow(3, i) ;

}

watch.stop();

System.out.println( "Time in Millis:" + watch.getTotalTimeMillis());

System.out.println( "Time in Secs:" + watch.getTotalTimeSeconds());

}

}

#3. Run main class

\*) IllegalStateException is throws if same method called twice in order

ex: start() and start() (or) stop() and stop().

===Ex#2===============================================

To calculate block/task specific time use method start(taskName) and also with constrcutor

StopWatch(id). Finally print output using method prettyPrint().

--code---

package in.nareshit.raghu.runner;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import org.springframework.util.StopWatch;

@Component

public class StopWatchTestRunner implements CommandLineRunner {

@Override

public void run(String... args) throws Exception {

StopWatch watch = new StopWatch("TestRunner-run()");

watch.start("LOOP#1"); //taskName

for (int i = 0; i < 10000000; i++) {

double result = Math.pow(i+1, i) + Math.pow(3, i) ;

}

watch.stop();

watch.start("LOOP#2"); //taskName

for (int i = 0; i < 200000000; i++) {

double result = Math.pow(i+1, i) + Math.pow(3, i) ;

}

watch.stop();

watch.start("LOOP#3"); //taskName

for (int i = 0; i < 30000000; i++) {

double result = Math.pow(i+1, i) + Math.pow(3, i) ;

}

watch.stop();

System.out.println(watch.prettyPrint());

}

}

Output:

StopWatch 'TestRunner-run()': running time = 7841992706 ns

---------------------------------------------

ns % Task name

---------------------------------------------

314647802 004% LOOP#1 SAVE OPR

6478829847 083% LOOP#2 EXPORT EXCEL

1048515057 013% LOOP#3 SEND EMAIL

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

**Scheduling using Spring Boot**

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

Executing a task in a loop over "PERIOD OF TIME" or "POINT OF TIME"

\*) PERIOD OF TIME : It indicates time gap (any start time/date)

Ex

6 Mins

24 hrs

4 Days

8 Months

\*) POINT OF TIME : It indicates exact date/time for start

Ex:

7AM

21st Sep 7AM

9AM 3rd Dec

7:50 PM

\*) Scheduling using Spring Boot:

#1. To activate Scheduling concept in Spring boot : @EnableScheduling (At main class)

#2. Define one class with method.

#3. On top of method apply @Scheduled annotation

#4. Choose one Scheduling Type(3)

-> fixedDelay (Period of time)

-> fixedRate (Period of time)

-> \*\*\* cron expression (Point of time\* / Period of time)

====Example=====

1. fixedDelay : This value is provided in MillSec. It will indicate "CALL OUR METHOD"

with GIVEN TIME GAP.

ex: fixedDelay=1000 . Then our method is called, once its execution is finished then

call 2nd time..etc

---Example Code--

package in.nareshit.raghu.service;

import java.util.Date;

import org.springframework.scheduling.annotation.Scheduled;

import org.springframework.stereotype.Component;

@Component

public class Messageservice {

@Scheduled(fixedDelay = 5000) //1000 mill sec = 1sec

public void printMsgA() {

System.out.println("HELLO " + new Date());

}

@Scheduled(fixedDelay = 1000) //1000 mill sec = 1sec

public void printMsgB() {

System.out.println("HI " + new Date());

}

}

\*)Note:

=> 1000 Mill Sec = 1 Sec.

=> Scheduler method never takes any parameter and returns nothing.

like public void <name>() { logic ... }

------------------------CORE JAVA TIME CONVERTER-----------------

TimeUnit(e) It is a Enum added in Java 1.5.

It is used to convert TIME/DAY values.

package in.nit.raghu;

import java.util.concurrent.TimeUnit;

public class Test {

public static void main(String[] args) {

// TimeUnit.FROM.toMethod(value);

//Days->Hrs

System.out.println(TimeUnit.DAYS.toHours(3));

//Days->Mins

System.out.println(TimeUnit.DAYS.toMinutes(3));

//Days->Sec

System.out.println(TimeUnit.DAYS.toSeconds(3));

System.out.println(TimeUnit.HOURS.toMinutes(2));

//Hrs->Days

System.out.println(TimeUnit.HOURS.toDays(48));

System.out.println(TimeUnit.MINUTES.toNanos(4));

//Sec->Mins

System.out.println(TimeUnit.SECONDS.toMinutes(120));

System.out.println(TimeUnit.SECONDS.toMillis(1));

System.out.println(TimeUnit.SECONDS.toMicros(1));

System.out.println(TimeUnit.SECONDS.toNanos(1));

}

}