## Spring Boot Scheduling

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

### Scheduling

**=>It is the process of executing given task or job either for 1 time or in a loop (for multiple times) based on given PERIOD OF TIME or POINT OF TIME**

**PERIOD OF TIME :: specifies the gap b/w two sucessive (back to back) executions of**

**the given task /job**

or

**POINT OF TIME :: executing the task at certain date, time and date and time (Mostly these are one time executions or once in a while executing tasks) =>The task/job that is enabled with scheduling will be executed automatically with out any human**

**intervention... and the taks/job executes for multiple times until the underlying application / server is stopped.**

**examples for PERIOD OF TIME Scheduled JOBS/TASKS**

**-> Every month payslip generation**

**-> Every month salary creding to account →**

**-> every month bank statement generation/**

**(The task/job executes in a loop having**

**certain time gap between succesive executions)**

**-> every day /every week/ every month sales report generation**

**-> sending emi remainders every month**

**-> sending insurance payment remainders every month and etc..**

**example for POINT OF TIME Scheduled JOBS/TASKS**

**These are repeately executing tasks**

**note:: It is like executing job/task at specific second or minute or hour or day**

**->Upload docs to certain website at specific night hour**

**->release of project at specific Date and time**

**of**

**->Converting certain form data (like csv data) to another form data (DB data)**

**at specific date and time**

**-> Releasing Entrance exam results at specific hour of sepcific date**

-> Starting certain moive ticket advance booking process on certain date and time

**-> starting certain product sales booking at certain date and time**

**-> Releasing youtube vedio to the channel at certain date and time.**

**and etc..**

**These are one time executing tasks..**

(or) Once in a while executing tasks

**=>In Jdk api we have TimerTask and Timer classes of java.util package to perform scheduling based**

**executions.**

we

**=> Once add "spring-boot-starter" dependency we automatically get scheduling support..**

**<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter**

**-->**

**<dependency>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-starter</artifactId>**

**<version> </version>**

**</dependency>**

**we need not to add this starter (jar++)**

**manually becoz it will come automatically**

**when we create any project as spring boot project**

**"spring-boot-starter" jar file/dependency gives AutoConfiguration + spring jars + logging + scheduling + yml processing+ .....**

**=> In Spring boot Apps we enable scheduling**

**a) By adding @EnableScheduling annotation on the top of main class /stater class along with @SpringBootApplication Annotation**

**b) In any spring bean class ( class with stereo type annotation) place b.methods having @Scheduled annotation**

**(This b.method should be designed with out params)**

**In @Scheduled**

**annotation,**

**we can specify**

**the**

**fixedRate**

**PERIOD**

**OF TIME and POINT OF TIME) (Best)**

**Example App**

**initialDelay (Specifices after starting app how much delay should be there to execute the sheduled job/task for 1st time)**

**fixedDelay | (supports only PERIOD OF TIME)**

**cron (Supports both**

**1) create spring Boot project with out adding any staraters**

**2) place @EnableScheduling on the top of main class**

**//main class**

**==========**

**package com.nt;**

E:\classcontent\spring\NTSPBMS715-RestfulServices-MicroServic Browse

BootSchedulingProj02

Service URL Name

https://start.spring.io BootSchedulingProj02

Use default location Location

Type:

Maven Project

Java Version:

17

✓ Packaging: Language:

Jar

Java

Group

nit

Artifact

Version

0.0.1-SNAPSHOT

**import java.util.Date;**

Description

Package

Demo project for Spring Boot com.nt

**import org.springframework.boot.SpringApplication;**

**import org.springframework.boot.autoconfigure.SpringBootApplication; import org.springframework.scheduling.annotation.EnableScheduling;**

**Enables Spring's scheduled task execution capability,**

(It makes the IOC container to search and locate @Scheduled enabled b.methods in the spring bean class)

**@SpringBootApplication**

**@EnableScheduling**

**public class SpringBootScheduling01Application {**

**public static void main(String[] args) {**

**SpringApplication.run(SpringBootScheduling01Application.class, args); System.out.println("App started at::"+new Date());**

}

}

**3) Develop spring bean having b.method enabled with Scheduling..**

package com.nt.report;

import java.util.Date;

import org.springframework.scheduling.annotation.Scheduled;

import org.springframework.stereotype.Component;

@Component("report")

public class ReportGenerator {

@Scheduled(initialDelay = 2000, fixed Delay = 3000) public void generateSalesReport() {

System.out.println("Sales Report on::"+new Date());

**note:: The method where @Scheduled is placed must be no param method becoz the IOC container**

}

}

**4) Run the Application..**

i

TIME

**can not pass those args while calling the method automatically**

**r**

**Sales**

**2024-02-16T18:23:29.813+05:30 INFO 21488 --- [ 2024-02-16T18:23:29.817+05:30 INFO 21488 --- [ 2024-02-16T18:23:30.485+05:30 INFO 21488 --- [ Application is started on::Fri Feb 16 18:23:30 IST 2024 Report on::Fri Feb 16 18:23:32 IST 2024 Sales Report on::Fri Feb 16 18:23:35 IST 2024 Sales Report on::Fri Feb 16 18:23:38 IST 2024 Sales Report on::Fri Feb 16 18:23:41 IST 2024 Sales Report on::Fri Feb 16 18:23:44 IST 2024 Sales Report on::Fri Feb 16 18:23:47 IST 2024 Sales Report on::Fri Feb 16 18:23:50 IST 2024 Sales Report on::Fri Feb 16 18:23:53 IST 2024**

//1000ms = 1sec

**The IOC container takes object for this**

**spring bean and calls this @Scheduled method repeately for multiple times having initialDelay and fixedDelay as specified.. until we stop the application ..**

**@Scheduled(fixedDelay = 3000)**

**public void generateReport(int start,int end) {**

main] t

**main] t**

**main] t**

}

**System.out.println(" Sales Report of ::"+new Date());**

**if we apply @Scheduled on the b.method that is having arg then**

**we get IllegalStateException as shown below**

**Caused by: java.lang.IllegalStateException: Encountered invalid @Scheduled method 'generateReport': Only no-arg methods may be annotated with @Scheduled**

**note:: if PERIOD of job/task is there in standalone App ... To stop that we need to stop the App (ctrl +c) note:: if PERIOD of job/task is there in web appliation... To stop that we need to stop the Server (ctrl +c)**

**TIME**

**if initialDay not specified the scheduled method trigger for execution along with App startup**

**@Scheduled(fixedDelay = 3000)**

**public void generateSalesReport() {**

**System.out.println("Sales Report on::"+new Date());**

**2024-02-16T18:28:48.148+05:30 INFO 5572 --- [ Sales Report on::Fri Feb 16 18:28:48 IST 2024 Application is started on::Fri Feb 16 18:28:48 IST 2024 Sales Report on::Fri Feb 16 18:28:51 IST 2024 Sales Report on::Fri Feb 16 18:28:54 IST 2024 Sales Report on::Fri Feb 16 18:28:57 IST 2024 Sales Report on::Fri Feb 16 18:29:00 IST 2024**

**=>we can specify fixed Delay time as stirng value as shown below**

**@Scheduled(fixed DelayString = "3000")**

**public void generateSalesReport() {**

**}**

I

***App startup time***

***and secheuld method trigging time is same***

**System.out.println("SalesReport on::"+new Date());**

**out**

***Q) can we place @Scheduled Annotation with parameters/attributes..***

***Ans) No ,we must place cron or fixed Delay or fixedRate parameters in the @Scheduled annotation otherwise exception will be raised***

**org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'report' defined in file [G:\Worskpaces\Spring\NTSPBMS714-BOOT\SpringBootScheduling01\target\classes\com\nt\report\ReportGenerator.class]: Initialization of bean failed; nested exception is java.lang.IllegalStateException: Encountered invalid @Scheduled method 'generateSales Report': Exactly one of the 'cron', 'fixed Delay(String)', or 'fixedRate(String)' attributes is required**

**fixedDealy**

**of**

**=>executes the task/job back to back having given time gap irrespective wheather task/job is completed fastly or slowly..**

**case1::**

**fixedDelay=3000 (3secs)**

**task1/job takes 15 secs time to complete**

**3secs gap/break**

**@Scheduled(fixedDelay = 3000)**

**task1/job1 execution for 15 secs**

**public void generateReport() {**

**System.out.println(" Sales Report of ::"+new Date()+"(start)");**

**try {**

**taks1/job1 execution form 15 secs**

**Thread.sleep(15000);**

3 secs gap/break

}

catch(Exception e) {

taks1/job1 execution form 15 secs

e.printStackTrace();

3 secs gap/break

}

System.out.println("end of salesReport"+new Date());

case2:: fixed Delay=3000 (3secs)

task1/job1 takes 1 sec time to complete

task1/job1 execution for 1sec

task1/job1 execution for 1sec

3 secs gap/break

3 secs gap/break

task1/job1 execution for 1sec

3 secs gap/break

example code

==========

```
@Component("report")
public class ReportGenerator {
@Scheduled(fixed DelayString = "3000")
public void generateSales Report() {
}
}
```

***fixedRate***

========

```
try {
Thread.sleep(5000);
}
catch (Exception e) { e.printStackTrace(); }
}
System.out.println("SalesReport on::"+new Date());
@Scheduled(fixedDelay = 3000)
public void generateReport() {
System.out.println(" Sales Report of ::"+new Date()+"(start)");
}
try {
```

**Thread.sleep(1000);**

**}**

**catch(Exception e) {**

**e.printStackTrace();**

}

**System.out.println("end of sales Report"+new Date());**

***When scheduling is enabled main thread represnets main app***

***and sub threds/child threads represents the scheduled jobs on 1 child thread per each scheduled job***

**=>specifies the max time that the task /job should take to complete the execution..**

***case1:: fixedRate: 10000 (10 secs)***

*task1/job1 is taking 5 secs to complete*

**@Scheduled(fixedRate =**

**= 1000**

***=>task1/job1 executin for 5secs***

**public void generateReport() {**

**System.out.println(" Sales Report of ::"+new Date()+"(start)");**

***10-5=5secs break/g ap***

**try {**

***=>task1/job1 execution for 5secs***

**Thread.sleep(5000);**

***10-5=5secs break/g ap***

}

**catch(Exception e) {**

***=>task1/job1 execution for 5secs***

**e.printStackTrace();**

10-5=5secs break/g ap

}

**System.out.println("end of sales Report"+new Date());**

**}**

***case2 :: fixedRate: 10000 (10 secs)***

***task1/job1 is taking15 secs to complete***

***=>task1/job1 executin for15secs***

***no break/gap/waiting***

*=>task1/job1 executin for15secs*

*no break/gap/waiting*

*=>task1/job1 executin for15secs*

***no break/gap/waiting***

()

**@Scheduled(fixedRate = 1000)**

**public void generateReport() {**

**System.out.println(" Sales Report of ::"+new Date()+"(start)");**

**try {**

**Thread.sleep(15000);**

}

**catch(Exception e) {**

**e.printStackTrace();**

}

**System.out.println("end of sales Report"+new Date());**

}

**note::**

**we can not place cron, fixedDelay and fixedRate attributes (either 3 or 2 attributes) at a time in the @Scheduled annotation**

**if we place we get Caused by: java.lang.IllegalStateException: Encountered invalid @Scheduled method 'generateReport': Exactly one of the 'cron', 'fixed Delay(String)', or 'fixedRate(String)' attributes is required**

**b/w**

**What is the difference fixed Delay and fixedRate attributes of @Scheduled annotation?**

**two**

**or**

***=>fixedDelay specifies the time gap/break time/wait time b/w succesive back to back executions So irrepspective of wheather task/job execution completed fastly or slowly that time gap between successive execution will be maintained. (It is like task assigment to employee of govt oranization i.e there will time gap b/w two successive jobs/tasks)***

**is**

***=>fixedReate specifies the max time that given to complete the execution of job/task .. if job/task execution is completed before the specified time the remaining will be used as the break time/gap time otherwise job/task executes back to back with out any gap/break time,***

**(It is like task assigment to employee of private oranization i.e task will be given with time frame) note: both fixedDelay and fixed Rate are there supporting only Period of Time style scheduling..**

**What are the differences among initialDelay, fixed Delay and fixedRate?**

**case1::**

**fixedDelay with initialDelay (PERIOD OF TIME)**

**task1/job1 execution time: 10 sec**

**initialDelay: 4secs**

**fixedDelay: 6secs**

**App**

**4sec taks1 break | task1 break | task1 | break | task1 | break**

**start**

**initial 10secs 6secs 10secs 6 secs**

**10secs 6 secs 10 secs 6secs**

**Delay**

**case2: fixedRate with initialDelay**

**where task execution time < fixedRate Time**

**task1/job1 execution time: 10 secs**

time

**initialDelay: 4secs**

**waiting time/breaktime :: fixedRate- task1 exeution time**

**(15secs 10 secs) = 5 secs**

**fixedRate: 15 secs**

App

**initial**

**Start**

**Delay**

**task1 break task1**

```
break task1 break
```

-

---

------

**4scs**

**10secs 5secs 10secs 5secs 10secs 5secs**

**case3: fixedRate with initialDealy where task execution time >=fixedRate time**

**task1/job1 execution time: 10 secs**

**initialDelay: 4secs**

**fixedRate: 8 secs**

**no waitingtime /breaktime**

**becoz fixedRate-task1 execution (8sec-10sec) gives**

**-2 (negetive numbers)**

**initial**

**App**

**task1 task1**

**task1**

**task1**

**task1**

**task1**

....

**(No break time)**

**delay**

**Start**

**4secs**

**10secs 10secs**

**10secs**

**10 secs 10secs**

**10secs**

```
@Component("report")
public class ReportGenerator {
//@Scheduled(initialDelay = 2000, fixed Delay = 3000)
//1000ms = 1sec
//@Scheduled(fixed Delay = 3000)
//@Scheduled(fixed DelayString = "3000")
@Scheduled(initialDelay =2000,fixedRate= 5000)
public void generateSalesReport() {
System.out.println("thread(task1) name ::"+Thread.currentThread().getName());
System.out.println("thread(task1) hashCode::"+Thread.currentThread().hashCode());
System.out.println("Report Data on"+new Date());
// main class runs with main thread
All
}
@Scheduled(fixed Delay =3000,initialDelay = 2000)
public void generateSalesReport1() {
System.out.println("thred (task2) name ::"+Thread.currentThread().getName());
System.out.println("thread(task2) hashCode::"+Thread.currentThread().hashCode());
System.out.println("Report Data1 on"+new Date());
}
```

**and scheduled tasks will trigger using same**

***child thread by defaut becoz thread pool size for scheduling***

-

**To increase that thread pool size use the following entry in application.properties**
**spring.task.scheduling.pool.size=20**

***(default is 1)***

**[ once we specify the thread pool size for scheduling.. the scheduled jobs/tasks will execute repeately using different threads of pool.. which improves the performence]**

***is "1" by default.***

**Spring boot scheduling with Cron Expressions**

**(*** very very important)**

===

***=> supports both PERIOD OF TIME and POINT OF TIME providing =>Inspired from unix/linux env.. way of date and time values =>Most regulary used scheduling process in spring /spring boot env..***

***=> The cron expression syntax is 6* syntax***

**=> Using fixed Delay and fixed Rate attributes of @Scheduled annotation we can bring the effect of only PERIOD OF TIME**

**=> using "cron" attribute of @Schduled Annotation we can bring the effect of both "PERIOD OF TIME" and "POINT OF TIME"**

***Week day name (SUN-SAT) 3 letter sytle or (0-7)***

Month of the year (1-12)

**0 or 7 represents sunday**

**date of the month → (1-31)**

***Hour of the Day***

*Minute of the hour*

***Second of the minute***

个

**(0-23)**

**(0-59)**

***(0-59)***

***Allowed symbols are :: */,? - LW @ #***

***In all versions***

***---> any/all /every***

**/ --> To specify PERIOD OF TIME**

***from sprng 5.3 onwords***

***-->To specify the possible list of values***

***---> To specify range of values***

**in**

***? ---> any (can be used only date and week day when month is specified)***

**in**

***L --->To specify Last days info (can be used only Date and Weeek day filelds)***

***W --> To specify Week Day Info (can be used only Date and Weeek day filelds) from spring***

***@ ---> To work with macros @yearly, @hourly, @monthly, @daily and etc.. 5.3 version***

***# --> As a combination symbol***

***To specify cron expression we need to use "cron" attribute: of***

***@Sheduled Annotation***

on

***Examples***

**POINT OF TIME (The given job/task executes on specific date and time)**

=====

**eg1: @Scheduled(cron = "15 **** *'*

*")**

***It is not executing task for every 15 secs ..***

***It is executing task on every 15 sec of every minute***

***9:01:15 secs***

***9:02:15 secs 9:03:15 secs***

***9:04:15 secs***

**@Component("report")**

**public class ReportGenerator {**

****

**@Scheduled(cron = "15 **

*")**

**public void generateSales Report() {**

**System.out.println("Sales Report Data::"+new Date());**

}

**eg2: @Scheduled (cron="0.09 ***")**

**execute**

**->The task will every day 9:00 am like**

**9:00am today**

**9:00 am tommorrow**

**9:00 am day after tommorrow**

**eg2 extension:**

==========

**every day evening 8:30:20 pm task should be executed**

**@Schduled(cron="20 30 20***")**

**Ex2 extesion1:: Task should be executed every hour 30 min 20 sec @Schedule(cron=" 20 30 ****")**

**What is meaning of this cron expression @Scheduled(cron="10 * 20 ***")**

**task executes every day night 8pm hour each min's 10 secs**

eg3: @Scheduled(cron="1 2 20** *")

**The task will execute every day at 8pm 02 minute 01 sec.**

**today at 8:02:01 pm**

**tommorrow at 8:02:01 pm**

**day after tommorrow at 8:02:01 pm**

**eg3-ext: @Scheduled(cron="******")**

**sec**

=> The given task will execute in every of

every

**miniute, every day and every hour**

**of every day and every month**

**20 hour means 12+8 hour (8pm)**

**eg3 extension:: The should be executed ever month 1st at 10 am @Scheduled(cron="0 0 10 1 **)**

**eg4: @Scheduled(cron="0 2 8,10 ***)**

**The task will execute every day at 8:02 am and 10:02 am**

**today at 8:02 am and 10:02 am**

**tommorrow at 8:02 am and 10:02 am**

**day after tommorrow at 8:02 am and 10:02 am**

**eg5: @Scheduled (cron="10 20 9-14 * *)**

**The task will execute every day at**

**9:20:10 am**

**10:20:10 am**

**11:20:10 am**

**12: 20:10 am 1:20:10 pm**

**2:20:10 pm**

**The task should be exeuted every minute at 0 sec**

**@Scheduled(cron="0"**

***")**

**List of**

**8,10 --> "," indicates possible values**

**Execute the task every hour at 8th minute and 10th minute**

**@Scheduled (cron="0 8,10 ****)**

**Execute the task from 10 th min to 20 min on 30th sec of every hour @Scheduled(30 10-20 ****)**

**Execute the task only leap lear feb 29 10 am**

**==========**

==========

**@Scheduled(cron="0 0 10 29 2 *")**

**eg6: Task should be executed every day 4pm Task should execute every at 7pm**

**@Scheduled(cron="0 0 16 ***)**

eg7 ::

**@Scheduled(" 1 2 56 **)**

**@Scheduled(cron="0 0 19 ***")**

**The task will execute every month 6 date 5:02:01 am**

**jan 6th 5:02:01 am**

**fed 6th 5:02:01 am**

**execute**

**Task should every month 10 th**

**18**

**@6: 30:20 pm @Sheduled(cron="20 30 10* *")**

**march 6th 5:02:01 am**

**eg8: @Scheduled (cron="* 18 8 ***")**

**=>Executes the give task in every second of**

**8:18 am min of every day**

**execute**

**Task should on every aug 15th 9:00:00 am**

**@Schedule(cron="0 0 9 15 08 *")**

**eg9: @Scheduled (cron=" 10 * 8 ***")**

**=> executes the given task in 10 sec of every miniute in the 8th hour**

**of every day**

**8:00:10 am**

The task should execute 20th sec of every

**minute in the 5pm hour of every day @Scheduled( cron=" 20 * 17 ** *")**

**8:01:10 am**

**8:02:10 am**

**8:03:10 am**

**The task should excute only on sun days at 10 am of every month @Scheduled("0 0 10** SUN")**

**0 or 7**

**@Scheduled(cron="0 0 0 1 1 0")**

**Task will execute 0:0:0 hours of jan 1st every year**

**only if it is sunday**

**Can u we execute the task only for 1 time using scheculing**

**Not Possible using @Scheduled annotation becoz all setup is given for the repeated executions**

**eg10: Execute the task at 10:00 am only on Sun-Thu days of the week**

**@Scheduled(cron="0 0 10* * SUN-THU)**

**eg11: Execute the task on every month 1st only if it is Sunday @Scheduled(cron="0 0 11 1* SUN")**

**eg12:: Execute the task every oct 5th only if it is tuesday**

**@Scheduled(cron="0 0 11 5 10 TUE ")**

**note:: In the cron expression we can use ? symbols in the place of * symbols for**

**day of month and for weekday**

**eg13:: Execute the task on any weekday of October month at 10 am @Scheduled(cron="0 0 10? 10 ?")**

**eg 14: execute the task on any day of the nov month having weekday SUN- TUE**

**@Scheduled(cron="0 0 11? 11 SUN-TUE")**