Prepared By - Nirmala Kumar Sahu

INDEX

Spring Boot Scheduling

Introduction

- Scheduling 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 between two successive (back-to-back) executions of the given task/ job.
- POINT OF TIME: executing the task at certain date, time, and both date & time.
- The task/ job that is enabled with scheduling will be executed automatically without any human intervention and the tasks/ job executes for multiple times until the underlying application/ server is stopped.

Examples for PERIOD OF TIME Scheduled JOBS/ TASKS:

Note: The task/ job executes in a loop having certain time gap between successive executions. These are repletely executing tasks.

- Every month pay slip generation.
- Every month salary crediting to account.
- Every month bank statement generation.
- Every day/ every week/ every month sales report generation.
- Sending EMI remainders every month.
- Sending insurance payment remainders every month.
  and etc.

Examples for POINT OF TIME Scheduled JOBS/ TASKS:

Note: It is like executing job/ task at specific second or minute or hour or day. These are one time executing tasks.

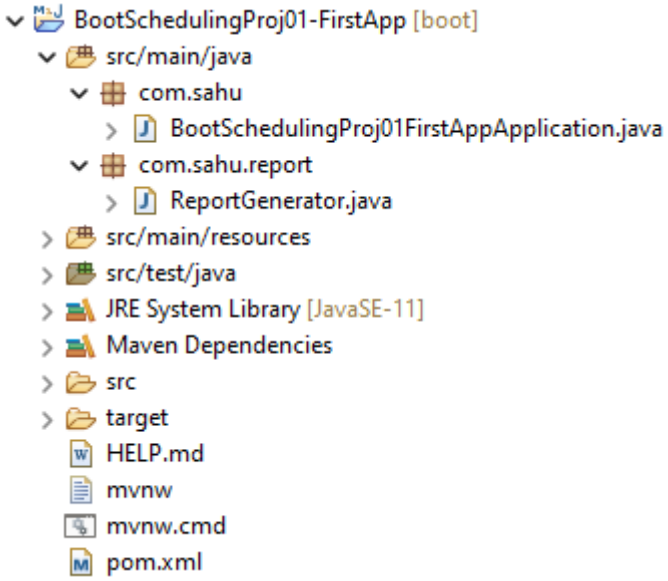
- Upload docs to certain website at specific night hour.
- Release of project at specific Date and time.
- Converting certain form data (like csv data) to another form data (DB data) at specific date and time.
- Releasing Entrance exam results at specific hour of specific date.
- Starting certain movie ticket advance booking process on certain date and time
- Starting certain product sales booking at certain date and time
- Releasing YouTube video to the channel at certain date and time.
  and etc.

- In JDK API we have TimerTask and Timer classes of java.util package to perform scheduling based executions.
- Once add "spring-boot-starter" dependency we automatically get scheduling support.
- "spring-boot-starter" jar file/ dependency gives AutoConfiguration + Spring jars + Logging + Scheduling + YML processing + many more.

In Spring Boot applications, we can 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 (annotation with stereo type annotation) place business methods having @Scheduled annotation.

Directory Structure of BootSchedulingProj01-FirstApp:



- Develop the above directory structure using Spring Starter Project option (Packaging: Jar) and create the package and class also.
- Need not to choose any starter during project creation.
- Then place the following code with in their respective files.

BootSchedulingProj01FirstAppApplication.java

```
@SpringBootApplication
@EnableScheduling
public class BootSchedulingProj01FirstAppApplication {

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

```
        SpringApplication.run(BootSchedulingProj01FirstAppApplication.class
, args);
                System.out.println("Application started at : "+new  Date());
        }

}
```

ReportGenerator.java

```
package com.sahu.report;

import java.util.Date;

import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

@Component("report")
public class ReportGenerator {

        @Scheduled(initialDelay = 2000, fixedDelay = 3000)
        public void generateSalesReport() {
                System.out.println("Sales Report on : "+new Date());
        }

}
```

In @Scheduled application, we can specify

- initialDelay: Specifics after starting app how much delay should be there to execute the scheduled job/ task for 1st time.
- fixedDelay & fixedRate: supports only PERIOD OF TIME.
- cron (Supports both PERID OF TIME and POINT OF TIME)

Note:

- ✓ The IoC container takes object for this Spring bean and calls this @Scheduled method repeatedly for multiple times having initialDelay and fixedDelay as specified until we stop the application.
- ✓ If PERIOD of job/ task is there in standalone application to stop that we need to stop the application (CTRL + c).
- ✓ If PERIOD of job/ task is there in web application to stop that we need to stop the server (CTRL + c).

- If initialDay not specified the scheduled method trigger for execution along with application startup. Application startup time and scheduled method trigging time is same.

ReportGenerator.java

```
@Component("report")
public class ReportGenerator {

	@Scheduled(fixedDelay = 3000)
	public void generateSalesReport() {
		System.out.println("Sales Report on : "+new Date());
	}

}
```

- We can specify fixedDelay time as String value as shown below
  @Scheduled(fixedDelayString = "3000")

Q. Can we place @Scheduled annotation with parameters/ attribute?
Ans. No, we must place cron or fixedDelay 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 [I:\JAVA\Workspace\Framework\SpringBoot\BootSchedulingProj01-FirstApp\target\classes\com\sahu\report\ReportGenerator.class]: Initialization of bean failed; nested exception is java.lang.IllegalStateException: Encountered invalid @Scheduled method 'generateSalesReport'*: Exactly one of the 'cron', 'fixedDelay(String)', or 'fixedRate(String)' attributes is required*

fixedDelay

- executes the task/ job back-to-back having given time gap irrespective whether task/ job is completed firstly or slowly.

Case 1: fixedDelay=3000 (3secs), task1/ job1 takes 15 secs time to complete
task 1/job 1 execution for 15 secs
3secs gap/ break
task 1/ job 1 execution form 15 secs
3 secs gap/break
....
.....

Case 2: fixedDelay=3000 (3secs), task 1/ job 1 takes 1 sec time to complete
task 1/ job 1 execution for 1 sec
3 secs gap/ break
task 1/ job 1 execution for 1 sec
3 secs gap/ break
….
….

ReportGenerator.java

```
@Component("report")
public class ReportGenerator {

	@Scheduled(fixedDelayString = "3000")
	public void generateSalesReport() {
		try {
			Thread.sleep(5000);
		}
		catch (Exception e) {
			e.printStackTrace();
		}
		System.out.println("Sales Report on : "+new Date());
	}

}
```

```
Application started at : Tue Feb 22 16:21:32 IST 2022
Sales Report on : Tue Feb 22 16:21:37 IST 2022
Sales Report on : Tue Feb 22 16:21:45 IST 2022
Sales Report on : Tue Feb 22 16:21:53 IST 2022
Sales Report on : Tue Feb 22 16:22:01 IST 2022
Sales Report on : Tue Feb 22 16:22:09 IST 2022
```

- When scheduling is enabled, main thread represents main app and sub threads/ child threads represents the scheduled jobs on 1 child thread per each scheduled job

fixedRate

- Specifies the max time that the task/ job should take to complete the execution.

Case 1: fixedRate: 10000 (10 secs), task 1/ job 1 is taking 5 secs to complete

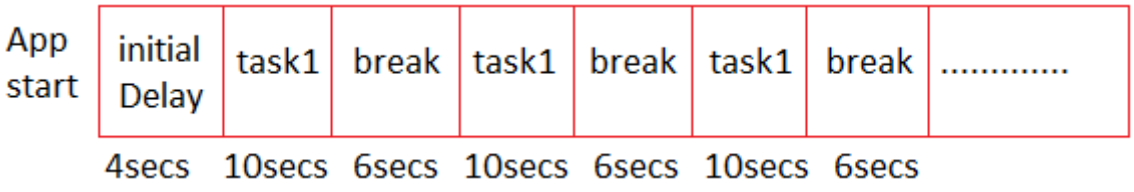
task 1/ job 1 execution for 5secs
10-5 = 5secs break/ gap
task 1/ job 1 execution for 5secs
10-5 = 5secs break/ gap
….
….

Case 2: fixedRate: 10000 (10 secs), task 1/ job 1 is taking15 secs to complete
task 1/ job 1 execution for 5secs
no break/ gap/ waiting
task 1/ job 1 execution for 5secs
no break/ gap/ waiting
….
….

Q. What is the difference fixedDelay and fixedRate?
Ans.

- fixedDelay specifies the time gap/ break time/ wait time between two successive back-to-back executions. So, irrespective of whether task/ job execution completed firstly or slowly that time gap between successive execution will be maintained.
- 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 without any gap/break time,

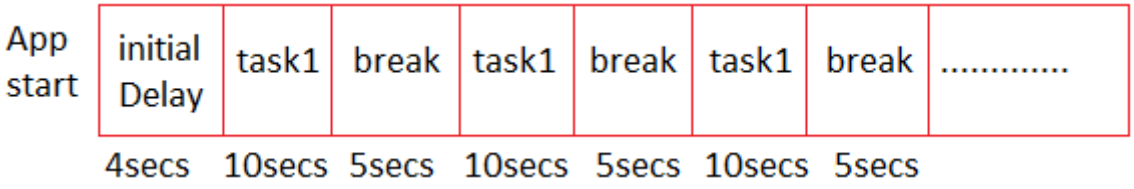
Q. What are the differences among initialDelay, fixedDelay and fixedRate?
Ans.
Case 1: fixedDelay with initialDelay (PERIOD OF TIME)
task 1/ job 1 execution time: 10 sec
initialDelay: 4secs
fixedDelay: 6secs



Case 2: fixedRate with initialDelay where task execution time < fixedRate
task1/ job1 execution time: 10 secs

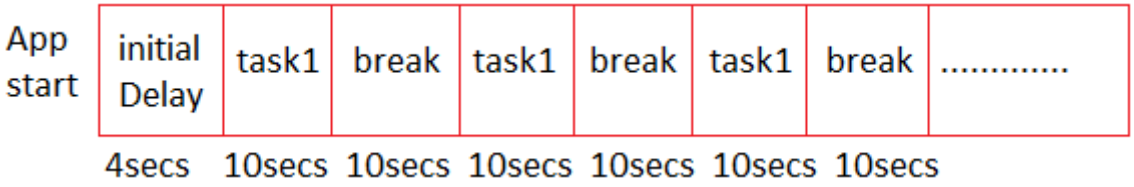
initialDelay: 4secs
fixedRate: 15 secs

waiting time/ breaktime: fixedRate – task1 execution time
(15secs - 10 secs) = 5 secs



Case 3: fixedRate with initialDealy where task execution time >= fixedRate time
task1/ job1 execution time: 10 secs
initialDelay: 4secs
fixedRate: 8 secs

no waiting time/ breaktime: because fixedRate – task1 execution
(8sec-10sec) = -2 (negative number)



Note: Main class runs with main thread and all scheduled tasks will trigger using same child thread by default because thread pool size for scheduling is "1" by default.

BootSchedulingProj01FirstAppApplication.java

```
public class BootSchedulingProj01FirstAppApplication {

	public static void main(String[] args) {

	SpringApplication.run(BootSchedulingProj01FirstAppApplication.class
, args);
		System.out.println("Thread Name -
"+Thread.currentThread().getName());
		System.out.println("Application started at : "+new  Date());
	}

}
```

ReportGenerator.java

```
@Component("report")
public class ReportGenerator {

	@Scheduled(initialDelay = 2000, fixedRate = 5000)
	public void generateSalesReport() {
		System.out.println("Task1 - Thread Name :
"+Thread.currentThread().getName());
		System.out.println("Task1 - Thread HashCode :
"+Thread.currentThread().hashCode());
		System.out.println("Task1 - Sales Report on : "+new Date());
	}

	@Scheduled(initialDelay = 2000, fixedDelay = 3000)
	public void generateSalesReport1() {
		System.out.println("Task2 - Thread Name :
"+Thread.currentThread().getName());
		System.out.println("Task2 - Thread HashCode :
"+Thread.currentThread().hashCode());
		System.out.println("Task2 - Sales Report on : "+new Date());
	}

}
```

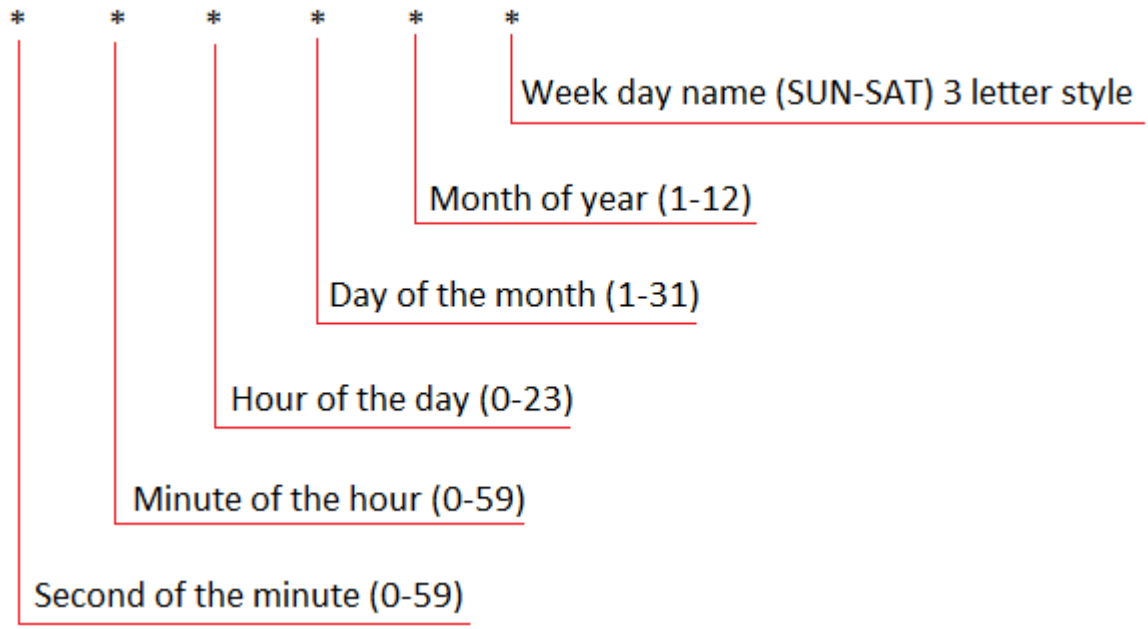
Note: To increase that thread pool size uses the following entry in

application.properties

```
spring.task.scheduling.pool.size=20
```

Spring Boot Scheduling with Cron Expressions

- Supports both PERIOD OF TIME and POINT OF TIME
- Inspired from Unix/ Linux environment way of providing date and time values.
- Most regularly used scheduling process in Spring/ Spring Boot environment
- The Cron expression syntax is 6 * syntax



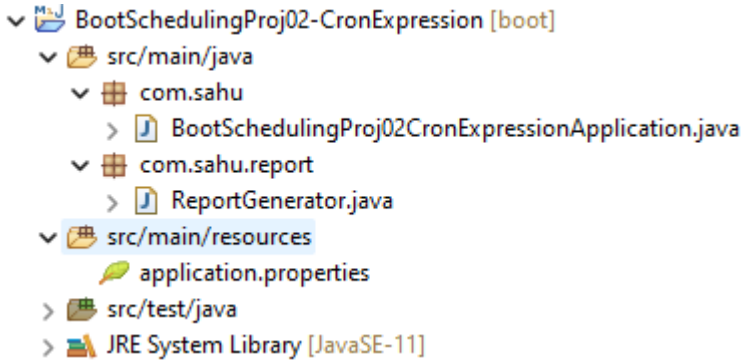
Allowed symbols are:

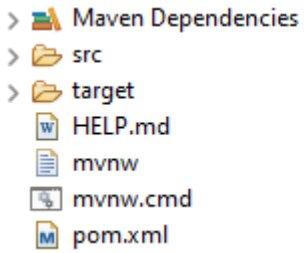
- In all versions: * / , ? –
- From Spring 5.3 onwards: L W @ #

- * any/ all /every
- / To specify PERIOD OF TIME
- , To specify the possible list of values
- - To specify range of values
- ? any (can be used only in date and week day)
- L To specify Last days info (can be used only Date and Week Day fields)
- W To specify Week Day Info (can be used only Date and Week Day fields)
- @ To work with macros @yearly, @hourly, @monthly, @daily and etc.
- # As a combination symbol

To specify Cron expression we need to use "cron" attributes of @Scheduled annotation.

Directory Structure of BootSchedulingProj02-CronExpression:





- Develop the above directory structure using Spring Starter Project option and create the package and class also.
- Need not to choose any starter during project creation.
- Then place the following code with in their respective files.

ReportGenerator.java

```
package com.sahu.report;

import java.util.Date;

import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

@Component("report")
public class ReportGenerator {

	@Scheduled(cron = "15 * * * * *")
	public void generateSalesReport() {
		System.out.println("Sales Report on : "+new Date());
	}

}
```

Cron expressions (POINT OF TIME)

E.g.,1: @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
  ....
  ......

E.g.,2: @Scheduled(cron = "0 0 9 * * *")

- The task will execute every day 9:00 am like
  9:00 am today
  9:00 am tomorrow
  9:00 am day after tomorrow
  ….
  ….

E.g.,3: @Scheduled(cron = "1 2 20 * * *")

- The task will execute every day at 8pm 02-minute 01 sec.
  today at 8:02:01 pm
  tomorrow at 8:02:01 pm
  day after tomorrow at 8:02:01 pm
  ….
  ….

E.g.,4: @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
  ….
  ….

Note: 8,10 here "," indicates possible values.

E.g.,5: @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
  01:20:10 pm
  02:20:10 pm

Note: After * symbols we cannot keep numbers.

E.g.,6: Task should be executed every day 4pm
@Scheduled(cron = "0 0 16 * * *")

E.g.,7: @Scheduled(cron = "1 2 5 6 * *")

- The task will execute every month 6 date 5:02:01 am
  Jan 6th 5:02:01 am
  Fed 6th am
  March 6th 5:02:01 am
  ….
  ….

E.g.,8: @Scheduled(cron = "0 0 6 9 5 SUN")

- Executes the given task every year May 9th @ 6:00 am if the May 9th week day is SUNDAY

E.g.,9: Execute task every year dec 31st 11:59:59 pm

@Scheduled(cron = "59 59 23 31 12 *")

Happy new year wishes before 1 sec

E.g.,10: Happy new year wishes on 0:0:0 am of Jan 1st

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

Or

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

E.g.,11: Wish me on teachers' day only if it is not Sunday (every year)

@Scheduled(cron = "0 0 0 5 9 MON-SAT")

task will be executing every year sept 5th 0:0:0 am if the sept 5th week day is MON to SAT.

E.g.,12: Execute the task every day of Jan month - every year

@Scheduled(cron = "0 0 0 1-31 1 *")

executes the at 0:0:0 am of every day in Jan month

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

@Scheduled(cron = "0 0 0 ? 1 ?") //valid

executes the task 0:0:0 am of every day in Jan month, this repeats every year.

@Scheduled(cron = "0 0 0 ? 1 *") //valid

? in date represents any/ every date

? in week day represents any/ every week day

E.g.,13: Given task should be executed every sun day of in March Month at 10:00:00 am

@Scheduled(cron = "0 0 10 ? MAR SUN")

Or

```
@Scheduled(cron = "0 0 10 1-31 MAR SUN")
```

E.g.,14: `@Scheduled(cron = "0 1 * 4 7 SUN")` //valid

E.g.,15: `@Scheduled(cron = "0 ? 1 6 9 *")` //invalid,
? symbol is not allowing in SEC place

E.g.,16: `@Scheduled(cron = "* * * ? JAN ?")` //valid

Note: ? meaning is same as * but can be used only in week day and date

E.g.,17: `@Scheduled(cron = "* * * 1-36 11 *")`
out of range only 1-28/31 are allowed

Cron expressions (PERIOD OF TIME)

- For this we can to place "/" symbol in every part of cron expression except in week day place (Last place).
- POINT OF TIME: Execution at specific month or date or hour or min or sec
- PERIOD OF TIME: Successive executions having time gap.
- Weekday range in cron Expression
  MON- SUN (or) 0-7 where 7 or 0 indicates Sunday

E.g.,1: `@Scheduled(cron = "0/20 * * * * *")`
- Execute the given task having 20 sec gaps

E.g.,2: `@Scheduled(cron = "10 0/15 * * * *")`
- Execute given task having 15 minutes gap at 10 sec

E.g.,3: `@Scheduled(cron = "20 0/2 10 * * * *")`
- Execute given task at the following time slots
  10:00:20 am
  10:02:20 am
  10:04:20 am

E.g.,4: `@Scheduled(cron = "30 0/1 16 * * *")`
- Execute the task every minute starting from 4pm at 30 sec.

E.g.,5: `@Scheduled(cron = "30 20/1 9 * * *")`

- Executes the task in the following timings
  - 9:20:30 am
  - 9:21:30 am
  - 9:22:30 am

E.g.,6: @Scheduled(cron = "0/20 0/30 10 * * *")

- Executes the task in the following timings
  - 10:00:00 am
  - 10:00:20 am
  - 10:00:40 am
  - 10:30:00 am
  - 10:30:30 am

E.g.,7: @Scheduled(cron = "4/5 9/10 10 * * *")

- Executes the task in the following timings
  - 10:09:04 sec
  - 10:09:09 sec
  - 10:09:14 sec
  - ....
  - 10:19:04 sec
  - 10:19:09 sec

- New Features of Cron expressions added from Spring 5.3 and Spring Boot 2.4
  a. Macros (@<....>)
  b. Last Days (L)
  c. Weekdays (W)
  d. Nth week day (<weekday>#<n>)

Macros

- Instead of six * Cron expression to repeat the task hourly or weekly or daily or monthly or yearly and etc. we can use macros directly.
- Expressions such as 0 0 * * * * are hard for humans to parse and are, therefore, hard to fix in case of bugs.
- To improve readability, Spring now supports the following macros, which represent commonly used sequences.

Macro	Meaning
@yearly (or @annually)	once a year (0 0 0 1 1 *)
@monthly	once a month (0 0 0 1 * *)

@weekly	once a week (0 0 0 * * 0)
@daily (or @midnight)	once a day (0 0 0 * * *)
@hourly	once an hour, (0 0 * * * *)

Last Days

- The day-of-month and day-of-week fields can contain a L character, which has a different meaning in each field. In the day-of-month field, L stands for the last day of the month.
- If followed by a negative offset (that is, L-n), it means nth-to-last day of the month.
- In the day-of-week field, L stands for the last day of the week. If prefixed by a number or three-letter name (dL or DDDL), it means the last day of week (d or DDD) in the month.

Cron Expression	Meaning
0 0 0 L * *	last day of the month at midnight
0 0 0 L-3 * *	third-to-last day of the month at midnight
0 0 0 * * 5L	last Friday of the month at midnight
0 0 0 * * THUL	last Thursday of the month at midnight

Second Friday of the Month

- The day-of-week field can be d#n (or DDD#n), which stands for the nth day of week d (or DDD) in the month.

Cron Expression	Meaning
0 0 0 ? * 5#2	the second Friday in the month at midnight
0 0 0 ? * MON#1	the first Monday in the month at midnight

Weekdays

- The day-of-month field can be nW, which stands for the nearest weekday to day of the month n. If n falls on Saturday, this yields the Friday before it.
- If n falls on Sunday, this yields the Monday after, which also happens if n is 1 and falls on a Saturday (that is: 1W stands for the first weekday of the month).

- If the day-of-month field is LW, it means the last weekday of the month.

Cron Expression	Meaning
0 0 0 1W * *	first weekday of the month at midnight
0 0 0 LW * *	last weekday of the month at midnight

@Scheduled(cron="0 11 7 LW * *") (setup: change to Nov 30 which is Tuesday)
@Scheduled(cron="0 15 7 IW * *") (setup :: change to Nov 1st which is Monday)

Note: While working NW and LW concepts manual date changes are not going to recognize.

Website link for Refer: QURTZ JOBS

---------- The END ----------