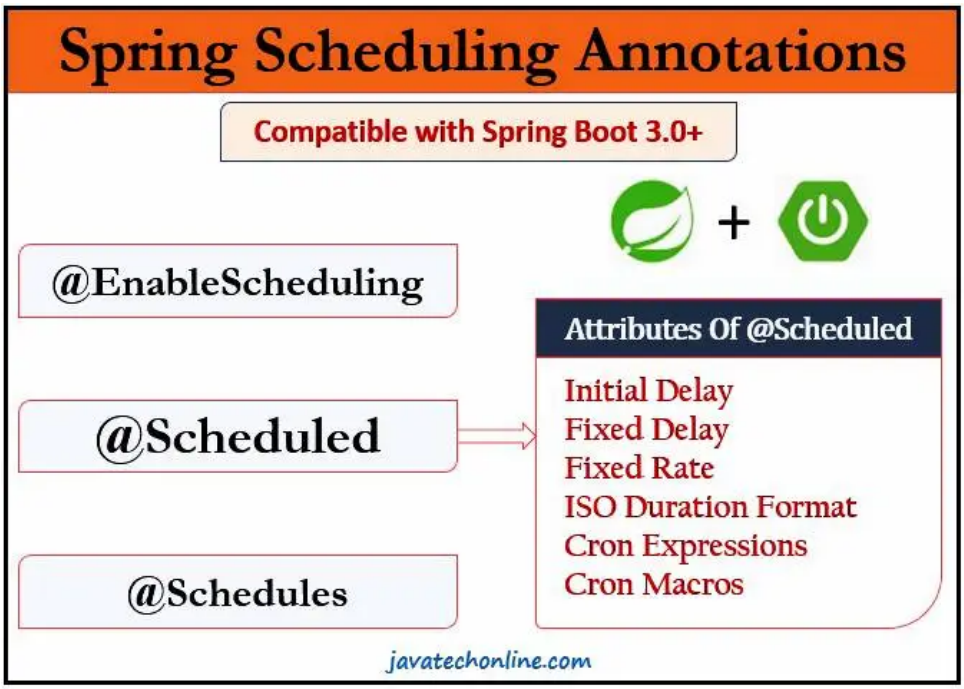
Making Java easy to learn
Java Technology and Beyond

Spring Scheduling Annotations With Examples



In this article, we will discuss on 'Spring Scheduling Annotations' with examples. Needless to say, Scheduling is an important part of a production grade application. Hence, learning of Spring Scheduling Annotations also becomes equally important. These annotations play a crucial role in creating an application in Spring Boot. If you want to learn all annotations which are generally used in a Spring Boot Project, kindly visit our article 'Spring Boot Annotations with Examples'. Let's discuss about 'Spring Scheduling Annotations' with examples here only.

Since scheduling is part of Spring Core, we don't need to add any specific dependency to get the features of it. Moreover, all the annotations of Spring Scheduling will also apply to Spring Boot Application. In order to have complete understanding on Scheduling in Spring Boot, you may visit Spring Boot Scheduler article.

Table of Contents

TOP

@EnableScheduling

Since scheduling is not enabled by default, we need to enable scheduling explicitly by adding the @EnableScheduling annotation before implementing any scheduled jobs. In order to enable Scheduling in our application, we need to apply *@EnableScheduling* in conjunction with *@Configuration*. We can also use it on our main class as it has *@Configuration* enabled by default. For example, below code demonstrates the use of annotation.

```
@EnableScheduling
@Configuration
public class MySchedulingConfig {
        ....
}
```

Please note that if you are using this annotation in your main class(having annotation *@SpringBootApplication*) of application, you don't need to apply *@Configuration,* as it is already available with*@SpringBootApplication*. Below code snippet illustrates the concept:

```
@EnableScheduling
@SpringBootApplication
public class SpringBootSchedulingApplication {

   public static void main(String[] args) {
      SpringApplication.run(SpringBootSchedulingApplication.class, args);
   }
}
```

@Scheduled

TOP

This is a method level annotation. If we want a method to execute periodically, we can annotate that method with *@Scheduled*. This annotation instructs Spring Container to execute the method in a loop as per the provided parameters until the application/server stops.

The method annotated with *@Scheduled* should not return any value, ie. It should have a void return type and even it should not have any parameter. Furthermore, it uses below concepts to support scheduling.

1) fixed Delay
2) fixed Rate
3) cron expression

@Scheduled can take one attribute from cron, fixedDelay, or fixedRate to determine the schedule of execution in various formats.

♦ **Note:** You can't write *@Scheduled* without providing any input as an attribute of it, otherwise Spring container will throw IllegalStateException: Encountered invalid *@Scheduled* method 'XYZ()': Exactly one of the 'cron', 'fixedDelay(String)', or 'fixedRate(String)' attributes is required.

@Scheduled with fixedDelay

A fixed delay represents the interval between the end of the previous job and the beginning of the new job.

```
@Scheduled(fixedDelay = 4000)
  // @Scheduled(fixedDelayString = "4000")
public void scheduledMethod() {
    System.out.println(" Scheduler with Fixed delay :" + new Date());
}
```

In case of fixedDelay, there is a delay of 4000 milliseconds(4 seconds) between the finish time of an execution of a task and the start time of the next execution of the task.

♥ **Note:** We can provide fixedDelay input in two ways : Integer and String (For example, look at the above code example)
Integer type : @Scheduled(fixedDelay = 4000)
String type : @Scheduled(fixedDelayString = "4000")

String value input also offers us to externalize the configuration by using Spring Expressions and storing them in properties files as shown in the example below:

TOP

```
@Scheduled(fixedDelayString = "${fixedDelay.input}")
@Scheduled(cron = "${cron.expression}")
```

Apart from that we can also utilize fixedDelayString and fixedRateString by specifying the time interval in the ISO duration format. We will discuss some of the examples in below sections.

@Scheduled with fixedDelay & initialDelay

We can opt to delay the first execution of the method by specifying the interval using the initialDelay attribute. The task will execute for the first time after the *initialDelay* value, and it will continue executing as per the *fixedDelay*.

```
@Scheduled(initialDelay = 5000, fixedDelay = 9000)
 // @Scheduled(initialDelayString = "5000" ,fixedDelayString = "9000")
public void scheduledMethod() {
    System.out.println(" Scheduler with Fixed delay and Initial Delay:" + new Date());
}
```

@Scheduled with fixedRate

The fixedRate attribute represents the interval for executing a job at a fixed interval of time. The execution time of the method is not considered when deciding when to start the next job.

```
@Scheduled(fixedRate = 4000)
// @Scheduled(fixedRateString = "4000")
public void scheduledMethod() {
    System.out.println(" Scheduler with Fixed Rate :" + new Date());
}
```

In case of fixedRate, the scheduled task will run at every 4000 milliseconds(4 seconds). It doesn't check for any previous executions of the task.

ISO Duration Format for fixedDelayString, initialDelayString & fixedRateString

In the above examples, we have provided the time intervals in milliseconds. It is fine to provide small values in milliseconds, but if values are large, representation in milliseconds will become difficu read. For example, the milliseconds representation for one hour will be 3600000. On the other h

TOP

we use ISO duration Format, it will be represented as 'PT01H'. Let's understand it with the help of below chart and an example:

```
@Scheduled(fixedDelayString = "PT20M")
public void scheduledMethod() {
    System.out.println(" Scheduler with Fixed delay:" + new Date());
}
```

Here, 'PT20M' represents 20 minutes, which is 1200000 in milliseconds. Similarly, we can represent initialDelayString & fixedRateString to specify a time interval in this format.

Moreover, we can also set these intervals as a property in our application.properties file. For example, the property named delay is set to 30 seconds in the duration format 'PT30S' as shown below.

```
@Scheduled(fixedDelayString = "${delay}")
```

```
delay=PT30S
```

@Scheduled with cron

TOP

Sometimes fixedDelay & fixedRate don't solve our purpose. In that case we should use cron expressions as it provides more flexible options. For example, below code demonstrates that the scheduled task will execute every year on Feb 14th 9:00:00 AM
if the given day(14th) is Sunday or Tuesday only.

```
@Scheduled(cron = "0 0 9 14 2 SUN,TUE")
public void scheduledMethod() {
    System.out.println("Hello cron Scheduler :" +new Date());
}
```

Moreoever, we can also use the zone attribute to modify this timezone as below. By default, Spring considers the server's local time zone for evaluation of the cron expression.

```
@Scheduled(cron = "0 0 9 14 2 SUN,TUE", zone = "America/New_York")
```

Furthermore, to get familiar with the variety of cron expressions, kindly visit a separate article on 'Spring Boot Scheduler Cron Expressions'.

CRON Macros

Spring also supports cron macros to represent most commonly used time intervals to improve code readability. Below macros are supported by Spring.

1) @hourly

2) @yearly

3) @monthly

4) @weekly

5) @daily

For example, below code snippet illustrates the usage of cron macros.

```
@Scheduled(cron = "@daily")
```

@Schedules

TOP

We can even configure multiple @Scheduled rules using *@Schedules* annotation. For example, below code demonstrates the concept:

```
@Schedules({
        @Scheduled(fixedRate = 4000),
        @Scheduled(cron = "0   0   0   14   2  *")
})
void getValentineDayNotification() { .... }
```

As shown in the example above, we have used two types of @Schedules rules.

In order to learn complete Scheduling with basics in Spring Boot, kindly visit our article on Spring Boot Scheduler.

Additionally, for a separate article specific to cron expressions with improvements in Spring 5.3, visit Spring Scheduling Cron Expressions & Improvements.

How to Schedule a Task/Job in Java Using Spring Boot Scheduler
November 4, 2020
In "Scheduling"

Spring Scheduling Cron Expression
January 16, 2023
In "java"

Spring Transaction Annotations With Examples
August 11, 2022
In "java"

Tagged @ebablescheduling @EnableScheduling @enablescheduling spring boot @Scheduled @scheduled in spring boot @scheduled spring @scheduled spring example @Schedules cron spring boot scheduled enable scheduling spring boot enablesheduling scheduled annotation Spring Boot - Scheduling spring boot enable scheduling spring boot enablescheduling spring boot scheduled transactional spring boot scheduler spring boot scheduling annotations spring cron expression spring enablescheduling spring scheduled cron format spring scheduled task programmatically spring scheduled transactional spring scheduler cron expressions spring scheduler example Spring Scheduling Annotations

‹ Previous article
Spring Security Annotations With Examples

Next article ›
Spring Transaction Annotations With Examples

TOP