Exécuter des tâches planifiées en arrière-plan dans Spring

View more Tutorials:

Tutoriels Spring Boot

Tutoriels Spring MVC

Parfois, dans une application, vous devez créer une tâche planifiée à exécuter en arrière-plan. Par exemple, créer un fichier **sitemaps**, envoyer un **email** périodiquement, ...

@Scheduled est une annotation utilisée pour configurer un planning (schedule), elle est annotée sur une méthode cette méthode sera exécutée selon le calendrier configuré par **@Scheduled**.

@Scheduled

```
1  public @interface Scheduled {
2
3      String cron() default "";
4
5      String zone() default "";
6
7      long fixedDelay() default -1;
8
9      String fixedDelayString() default "";
10
11     long fixedRate() default -1;
12
13     String fixedRateString() default "";
14
15     long initialDelay() default -1;
16
17     String initialDelayString() default "";
18
19 }
```

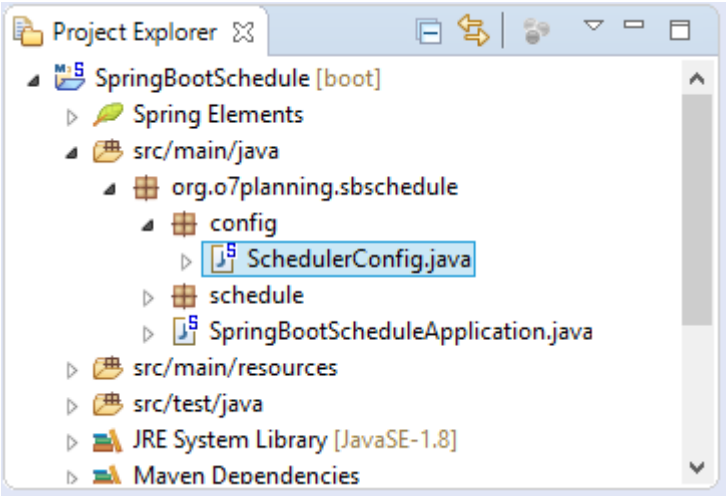
Attribut	Description
cron	Une expression de type cron, étendant la définition habituelle de l' **UN*X**, elle comprend 6 champs "seconde, minute, heure, le jour du mois, le mois et le jour de la semaine". Elle indique un plan compliqué. (Voir plus au document ci-dessous). • @return une expression qui peut être analysée dans un plan de cron
zone	Un fuseau horaire pour lequel l'expression **cron** sera résolue. Par défaut, cet attribut est la chaîne vide (c'est-à-dire que le fuseau horaire local du serveur sera utilisé).
fixedDelay	Exécutez la méthode annotée (annotated) avec une période fixe en millisecondes entre la fin de la dernière invocation et le début de la prochaine. • @return le délai en millisecondes.
fixedDelayString	Exécutez la méthode annotée (annotated) avec une période fixe en millisecondes entre la fin de la dernière invocation et le début de la prochaine.. • @return le délai en millisecondes en tant que valeur de chaîne, par exemple un espace réservé.
	Exécutez la méthode annotée (annotated) avec une période fixe en millisecondes entre les invocations.

fixedRateString	• @return la période en millisecondes en tant que valeur de chaîne, par ex. un espace réservé.
initialDelay	Le nombre de millisecondes à retarder avant la première exécution d'une tâche **fixedRate()** ou **fixedDelay()**. • @return le délai initial en millisecondes
initialDelayString	Nombre de millisecondes à retarder avant la première exécution d'une tâche **fixedRate()** ou **fixedDelay()**

- @return le délai initial en millisecondes en tant que valeur de chaîne, par ex. un espace réservé.

2- @EnableScheduling Annotation

Pour pouvoir exécuter des tâches d'arrière-plan dans une application **Spring**, vous devez la configurer pour activer la planification en utilisant **@EnableScheduling**.

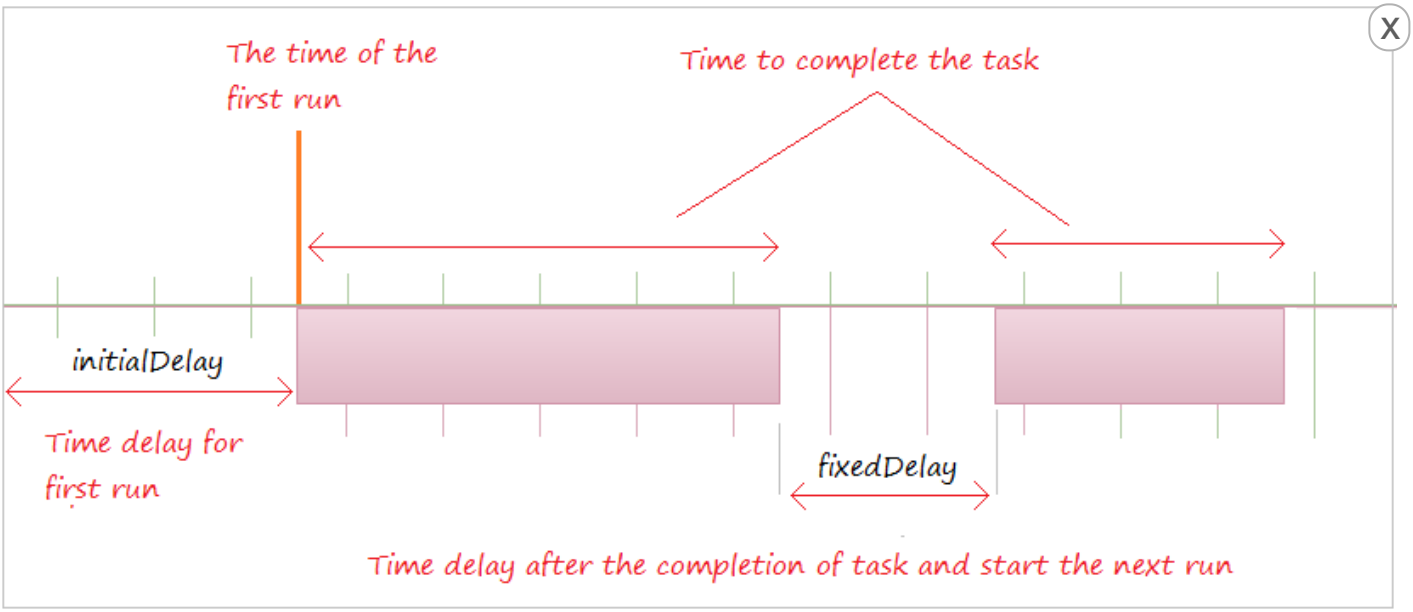


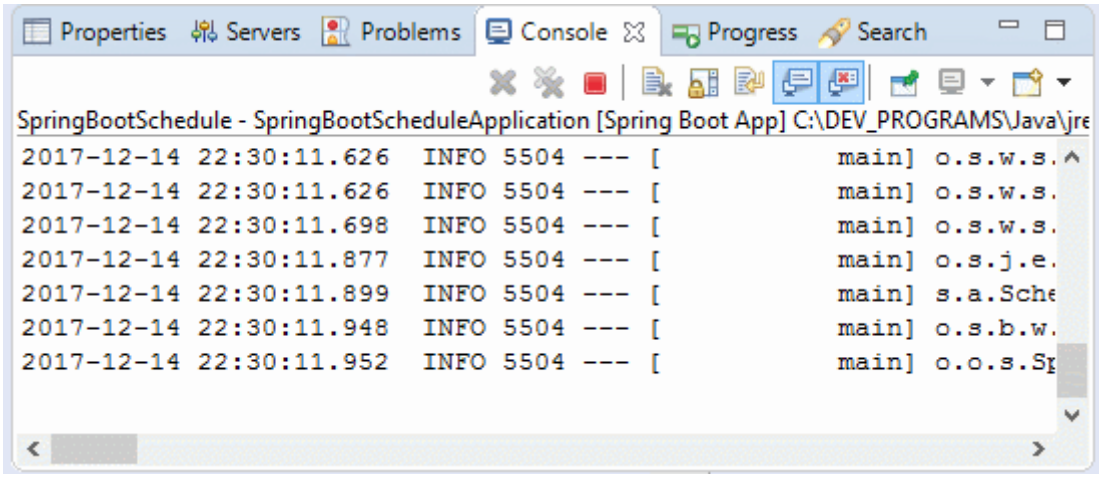
SchedulerConfig.java

```
1  package org.o7planning.sbschedule.config;
2
3  import org.springframework.context.annotation.Configuration;
4  import org.springframework.scheduling.annotation.EnableScheduling;
5
6  @Configuration
7  @EnableScheduling
8  public class SchedulerConfig {
9
10    // Declaring the beans are related to the schedule here if necessary.
11 }
```

3- fixedDelay & fixedRate

C'est l'exemple le plus simple, utilisant **@Schedule** avec l'attribut ***fixedDelay***. Dans cet exemple, la tâche imprimera l'heure actuelle **Console** et lorsque la tâche est terminé, elle arrêtera ***fixedDelay*** le seconde avant qu'elle effectue la tâche à nouveau.





WriteCurrentTimeSchedule.java

```
1  package org.o7planning.sbschedule.schedule;
2
3  import java.text.DateFormat;
4  import java.text.SimpleDateFormat;
5  import java.util.Date;
6
7  import org.springframework.scheduling.annotation.Scheduled;
8  import org.springframework.stereotype.Component;
9
10 @Component
11 public class WriteCurrentTimeSchedule {
12
13    private static final DateFormat df = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss sss");
```



```
22
23       String nowString = df.format(now);
24
25       System.out.println("Now is: "+ nowString);
26
27    }
28
29 }
```

Vous pouvez configurer le temps de retard pour la première exécution (**initialDelay**), après l'application **start**, la tâche est effectuée uniquement lorsque le délai est en cours.

```
// miliseconds.
@Scheduled(fixedDelay = 2 * 1000, initialDelay = 30 * 1000)
public void writeCurrentTime() {

   // Do something ..

}
```

Comparer fixedDelay avec fixedRate

Vous ne pouvez utiliser que **fixedDelay** ou **fixedRate** dans l'annotation **@Schedule**, mais vous ne pouvez pas utiliser les deux simultanément.

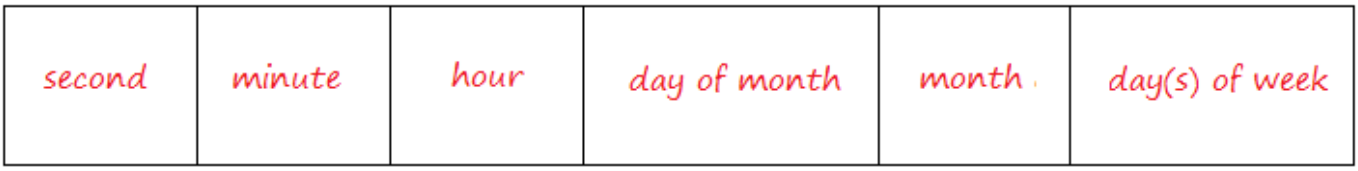
- *fixedDelay*est la période de repos lorsque la tâche est terminée. Après le temps de repos, il effectuera la prochaine tâche.
- *fixedRate* est la période qui se situe entre l'heure de début d'exécution de la tâche précédente et l'heure de début pour effectuer la tâche suivante, elle ne dépend pas si la tâche précédente est terminée ou non.

4- L'expression cron et zone

@Schedule vous permet de configurer des programmes complexes, par exemple, effectuer des tâches toutes les 12 heures lundi. Pour configurer les programmes complexes, vous pouvez utiliser l'attribut **cron**.

L'expression **cron** est représentée par six champs:

```
second, minute, hour, day of month, month, day(s) of week
```



Exemple:

```
"0 0 * * * *" // the top of every hour of every day.
```



```
"0 0 0 25 12 ?" // every Christmas Day at midnight
```

Description:

Modèle	Signification
*	signifie correspondre à quelques
*/X	signifie "Tous les X"

?	("Aucune valeur spécifique") - utile lorsque vous devez spécifier quelque chose dans l'un des deux champs dans lesquels le caractère est autorisé, mais pas l'autre. Par exemple, si je veux le 10ème jour dans un mois mais je ne concerne pas il sera quel jour dans la semaine, je dois mettre "10" au champ *day-of-month*, et mettre "?" au champ *day-of-week*.

"0 0 0 25 12 ?":

À 12 minuit (0 heure) le jour de Noël (le 25 décembre), ça ne vous dérange pas à quel jour il est.

L'attribut **zone** spécifiait le fuseau horaire. La valeur par défaut de **zone** est vide, ce qui signifie utiliser le fuseau horaire du **server**.

```
@Scheduled(cron="0 1 1 * * *", zone="Europe/Istanbul")

public void doScheduledWork() {
    // Do something here
```


Internal Name	External User Visible Name
Pacific/Pago_Pago	(-11:00) Pago Pago
Pacific/Honolulu	(-10:00) Hawaii
America/Anchorage	(-09:00) Alaska
America/Vancouver	(-08:00) Canada Pacific Time
America/Los_Angeles	(-08:00) US Pacific Time

Internal Name	External User Visible Name
America/Tijuana	(-08:00) Tijuana
America/Edmonton	(-07:00) Canada Mountain Time
America/Denver	(-07:00) US Mountain Time
America/Phoenix	(-07:00) Arizona
America/Mazatlan	(-07:00) Mazatlan
America/Winnipeg	(-06:00) Canada Central Time
America/Regina	(-06:00) Saskatchewan
America/Chicago	(-06:00) US Central Time
America/Mexico_City	(-06:00) Mexico City
America/Guatemala	(-06:00) Guatemala
America/El_Salvador	(-06:00) El Salvador
America/Managua	(-06:00) Managua
America/Costa_Rica	(-06:00) Costa Rica
America/Montreal	(-05:00) Canada Eastern Time
America/New_York	(-05:00) US Eastern Time
America/Indianapolis	(-05:00) East Indiana
America/Panama	(-05:00) Panama
America/Bogota	(-05:00) Bogota
America/Lima	(-05:00) Lima
America/Halifax	(-04:00) Canada Atlantic Time
America/Puerto_Rico	(-04:00) Puerto Rico
America/Caracas	(-04:00) Caracas
America/Santiago	(-04:00) Santiago
America/St_Johns	(-03:30) Newfoundland
America/Sao_Paulo	(-03:00) Sao Paulo
Atlantic/Azores	(-01:00) Azores
Etc./UTC	(00:00) Universal Time
UTC	(00:00) Universal Time
Atlantic/Reykjavik	(00:00) Reykjavik
Europe/Dublin	(00:00) Dublin
Europe/London	(00:00) London
Europe/Lisbon	(00:00) Lisbon
Africa/Casablanca	(00:00) Casablanca
Africa/Nouakchott	(00:00) Nouakchott
Europe/Oslo	(+01:00) Oslo
Europe/Stockholm	(+01:00) Stockholm
Europe/Copenhagen	(+01:00) Copenhagen

Internal Name	External User Visible Name
Europe/Berlin	(+01:00) Berlin
Europe/Amsterdam	(+01:00) Amsterdam
Europe/Brussels	(+01:00) Brussels
Europe/Luxembourg	(+01:00) Luxembourg
Europe/Paris	(+01:00) Paris
Europe/Zurich	(+01:00) Zurich
Europe/Madrid	(+01:00) Madrid
Europe/Rome	(+01:00) Rome
Africa/Algiers	(+01:00) Algiers
Africa/Tunis	(+01:00) Tunis
Europe/Warsaw	(+01:00) Warsaw
Europe/Prague	(+01:00) Prague Bratislava
Europe/Vienna	(+01:00) Vienna
Europe/Budapest	(+01:00) Budapest
Europe/Sofia	(+02:00) Sofia
Europe/Istanbul	(+02:00) Istanbul
Europe/Athens	(+02:00) Athens
Asia/Nicosia	(+02:00) Nicosia
Asia/Beirut	(+02:00) Beirut
Asia/Damascus	(+02:00) Damascus
Asia/Jerusalem	(+02:00) Jerusalem
Asia/Amman	(+02:00) Amman
Africa/Tripoli	(+02:00) Tripoli
Africa/Cairo	(+02:00) Cairo
Africa/Johannesburg	(+02:00) Johannesburg
Europe/Moscow	(+03:00) Moscow
Asia/Baghdad	(+03:00) Baghdad
Asia/Kuwait	(+03:00) Kuwait
Asia/Riyadh	(+03:00) Riyadh
Asia/Bahrain	(+03:00) Bahrain
Asia/Qatar	(+03:00) Qatar
Asia/Aden	(+03:00) Aden
Africa/Khartoum	(+03:00) Khartoum
Africa/Djibouti	(+03:00) Djibouti
Africa/Mogadishu	(+03:00) Mogadishu
Asia/Dubai	(+04:00) Dubai
Asia/Muscat	(+04:00) Muscat

Internal Name	External User Visible Name
Asia/Yekaterinburg	(+05:00) Yekaterinburg
Asia/Tashkent	(+05:00) Tashkent
Asia/Calcutta	(+05:30) India
Asia/Novosibirsk	(+06:00) Novosibirsk
Asia/Almaty	(+06:00) Almaty
Asia/Dacca	(+06:00) Dacca
Asia/Krasnoyarsk	(+07:00) Krasnoyarsk
Asia/Bangkok	(+07:00) Bangkok
Asia/Saigon	(+07:00) Vietnam
Asia/Jakarta	(+07:00) Jakarta
Asia/Irkutsk	(+08:00) Irkutsk
Asia/Shanghai	(+08:00) Beijing, Shanghai
Asia/Hong_Kong	(+08:00) Hong Kong
Asia/Taipei	(+08:00) Taipei
Asia/Kuala_Lumpur	(+08:00) Kuala Lumpur
Asia/Singapore	(+08:00) Singapore
Australia/Perth	(+08:00) Perth
Asia/Yakutsk	(+09:00) Yakutsk
Asia/Seoul	(+09:00) Seoul
Asia/Tokyo	(+09:00) Tokyo
Australia/Darwin	(+09:30) Darwin
Australia/Adelaide	(+09:30) Adelaide
Asia/Vladivostok	(+10:00) Vladivostok
Australia/Brisbane	(+10:00) Brisbane
Australia/Sydney	(+10:00) Sydney Canberra
Australia/Hobart	(+10:00) Hobart
Asia/Magadan	(+11:00) Magadan
Asia/Kamchatka	(+12:00) Kamchatka
Pacific/Auckland	(+12:00) Auckland

Dans l'exemple ci-dessous, la tâche se déroule à 1h20' lundi, mardi, mercredi et jeudi.

```
1 "0 20 1 * * MON-THU"
```

CronSchedule.java

```java
package org.o7planning.sbschedule.schedule;

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

@Component
public class CronSchedule {

    @Scheduled(cron = "0 20 1 * * MON-THU")
    public void doSomething() {

        System.out.println("Do some thing");

    }

}
```

View more Tutorials:

Tutoriels Spring Boot

Tutoriels Spring MVC

Peut-être que vous êtes intéressé

Voici des leçons en ligne à part du site web o7planning que nous recommandons. La liste comprend des leçons en ligne et celles en promo.

- Udemy Spring Framework 5: Beginner to Guru
- Udemy Learning Path: Spring and Spring Boot Projects
- Udemy Master Java Web Services and RESTful API with Spring Boot
- Udemy Building a Web Application with Spring and Angular
- Udemy Mastering Thymeleaf with Spring Boot
- Udemy Spring Boot and Java Development with IntelliJ IDEA
- Udemy Mastering Micro Services Using Java Spring Boot
- Udemy Essentials of Spring 5.0 for Developers
- Udemy Learning Web Application with Spring 5 and Angular 2
- Udemy Java :Spring and Hibernate Restful web service crud
- Udemy Learning Spring Boot