Planification des tâches dans Spring

La planification des tâches dans Spring est une exigence courante pour automatiser des tâches répétitives, telles que l'envoi d'e-mails, la génération de rapports ou la réalisation de la maintenance du système. Spring fournit une fonctionnalité de planification de tâches puissante et facile à utiliser grâce à son annotation `@Scheduled` et au planificateur de tâches Spring. Ci-dessous, je vais vous guider à travers les étapes pour la configurer et expliquer les concepts clés.

1. Activer la planification dans votre application Spring

Pour utiliser la planification, vous devez l'activer dans votre application Spring. Cela se fait en ajoutant l'annotation `@EnableScheduling` à une classe de configuration.

```
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.annotation.EnableScheduling;

@Configuration
@EnableScheduling
public class SchedulerConfig {
    // La classe de configuration peut être vide sauf si vous avez besoin de paramètres de planificateur pers
}
```

Cela indique à Spring de rechercher les méthodes annotées avec `@Scheduled` et de les exécuter selon leurs horaires définis.

---

2. Créer une tâche à planifier

Vous pouvez définir une méthode dans n'importe quel bean géré par Spring (comme un `@Component` ou `@Service`) et l'annoter avec `@Scheduled`. Voici un exemple :

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

@Component
public class MyScheduledTasks {

    // S'exécute toutes les 5 secondes
    @Scheduled(fixedRate = 5000)
    public void performTask() {
```

```
        System.out.println("Tâche exécutée à : " + System.currentTimeMillis());
    }
}
```

Dans cet exemple : - `@Component` rend la classe un bean Spring. - `@Scheduled(fixedRate = 5000)` exécute la méthode toutes les 5 secondes (5000 millisecondes).

---

3. Types d'options de planification

Spring propose plusieurs façons de définir quand une tâche doit s'exécuter :

a) Taux fixe

- Exécute la tâche à un intervalle fixe, indépendamment de la durée de la tâche.
- Exemple : `@Scheduled(fixedRate = 5000)` (toutes les 5 secondes).

b) Délai fixe

- Exécute la tâche avec un délai fixe entre la fin d'une exécution et le début de la suivante.
- Exemple : `@Scheduled(fixedDelay = 5000)` (5 secondes après la fin de la tâche précédente).

c) Expression Cron

- Utilise une syntaxe similaire à cron pour des horaires plus complexes (par exemple, "chaque jour ouvrable à 9h").
- Exemple : `@Scheduled(cron = "0 0 9 * * MON-FRI")`.

d) Délai initial

- Retarde la première exécution de la tâche. Combinez-le avec `fixedRate` ou `fixedDelay`.
- Exemple : `@Scheduled(fixedRate = 5000, initialDelay = 10000)` (démarre après 10 secondes, puis s'exécute toutes les 5 secondes).

---

4. Bases de la syntaxe Cron

Si vous utilisez cron, voici une référence rapide : - Format : `second minute hour day-of-month month day-of-week` - Exemple : `@Scheduled(cron = "0 15 10 * * ?")` (s'exécute à 10h15 tous les jours).

Champ	Valeurs autorisées
Seconde	0-59
Minute	0-59
Heure	0-23
Jour du mois	1-31
Mois	1-12 ou JAN-DEC
Jour de la semaine	0-7 ou SUN-SAT (0 ou 7 est dimanche)

Des caractères spéciaux comme $*$ (tous), , (liste), – (plage), / (incrément) peuvent être utilisés.

---

5. Dépendances

Si vous utilisez Spring Boot, la fonctionnalité de planification est incluse par défaut dans la dépendance `spring-boot-starter`. Pour un projet Spring non-Boot, assurez-vous d'avoir :

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>6.1.3</version> <!-- Utilisez la dernière version -->
</dependency>
```

---

6. Exemple complet avec Spring Boot

Voici un exemple complet de Spring Boot :

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.annotation.EnableScheduling;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;
```

```java
@SpringBootApplication
@EnableScheduling
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}

@Component
class ScheduledTasks {

    @Scheduled(fixedRate = 5000)
    public void runEveryFiveSeconds() {
        System.out.println("S'exécute toutes les 5 secondes : " + System.currentTimeMillis());
    }

    @Scheduled(cron = "0 */1 * * * *")
    public void runEveryMinute() {
        System.out.println("S'exécute toutes les minutes : " + System.currentTimeMillis());
    }
}
```

---

7. Personnalisation du planificateur (facultatif)

Par défaut, Spring utilise un planificateur monothread. Si vous avez besoin d'exécuter des tâches de manière concurrente, vous pouvez configurer un `TaskScheduler` personnalisé :

```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.concurrent.ThreadPoolTaskScheduler;

@Configuration
public class SchedulerConfig {

    @Bean
    public ThreadPoolTaskScheduler taskScheduler() {
        ThreadPoolTaskScheduler scheduler = new ThreadPoolTaskScheduler();
        scheduler.setPoolSize(10); // Nombre de threads
```

```
        scheduler.setThreadNamePrefix("my-scheduler-");
        return scheduler;
    }
}
```

Cela permet à plusieurs tâches de s’exécuter en parallèle.

---

8. Tester vos tâches planifiées

- Exécutez votre application et vérifiez les journaux de la console pour vous assurer que les tâches s’exécutent comme prévu.
- Utilisez des intervalles plus courts (par exemple, toutes les 2 secondes) pendant les tests pour voir les résultats rapidement.