Programación de Tareas en Spring

Programar tareas en Spring es un requisito común para automatizar trabajos repetitivos, como enviar correos electrónicos, generar informes o realizar mantenimiento del sistema. Spring proporciona una característica de programación de tareas poderosa y fácil de usar a través de su anotación `@Scheduled` y el Programador de Tareas de Spring. A continuación, te guiaré a través de los pasos para configurarlo y explicaré los conceptos clave.

1. Habilitar la Programación en tu Aplicación Spring

Para usar la programación, debes habilitarla en tu aplicación Spring. Esto se hace agregando la anotación `@EnableScheduling` a una clase de configuración.

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

@Configuration
@EnableScheduling
public class SchedulerConfig {
    // La clase de configuración puede estar vacía a menos que necesites configuraciones de programador perso
}
```

Esto le indica a Spring que busque métodos anotados con `@Scheduled` y los ejecute según sus horarios definidos.

---

2. Crear una Tarea para Programar

Puedes definir un método en cualquier bean gestionado por Spring (como un `@Component` o `@Service`) y anotarlo con `@Scheduled`. Aquí tienes un ejemplo:

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

@Component
public class MyScheduledTasks {

    // Se ejecuta cada 5 segundos
    @Scheduled(fixedRate = 5000)
    public void performTask() {
```

```
        System.out.println("Tarea ejecutada a: " + System.currentTimeMillis());
    }
}
```

En este ejemplo: - `@Component` hace que la clase sea un bean de Spring. - `@Scheduled(fixedRate = 5000)` ejecuta el método cada 5 segundos (5000 milisegundos).

---

3. Tipos de Opciones de Programación

Spring ofrece varias formas de definir cuándo debe ejecutarse una tarea:

a) Tasa Fija

- Ejecuta la tarea a intervalos fijos, independientemente de cuánto tiempo tome la tarea.
- Ejemplo: `@Scheduled(fixedRate = 5000)` (cada 5 segundos).

b) Retraso Fijo

- Ejecuta la tarea con un retraso fijo entre el final de una ejecución y el inicio de la siguiente.
- Ejemplo: `@Scheduled(fixedDelay = 5000)` (5 segundos después de que termine la tarea anterior).

c) Expresión Cron

- Usa una sintaxis similar a cron para horarios más complejos (por ejemplo, “todos los días laborables a las 9 AM”).
- Ejemplo: `@Scheduled(cron = "0 0 9 * * MON-FRI")`.

d) Retraso Inicial

- Retrasa la primera ejecución de la tarea. Combínalo con `fixedRate` o `fixedDelay`.
- Ejemplo: `@Scheduled(fixedRate = 5000, initialDelay = 10000)` (comienza después de 10 segundos, luego se ejecuta cada 5 segundos).

---

4. Básicos de la Sintaxis Cron

Si usas cron, aquí tienes una referencia rápida: - Formato: `segundo minuto hora día-del-mes mes día-de-la-semana` - Ejemplo: `@Scheduled(cron = "0 15 10 * * ?")` (se ejecuta a las 10:15 AM todos los días).

Campo	Valores Permitidos
Segundo	0-59
Minuto	0-59
Hora	0-23
Día del Mes	1-31
Mes	1-12 o ENE-DIC
Día de la Semana	0-7 o DOM-SAB (0 o 7 es domingo)

Puedes usar caracteres especiales como * (todos), , (lista), – (rango), / (incremento).

---

5. Dependencias

Si estás usando Spring Boot, la característica de programación está incluida por defecto en la dependencia `spring-boot-starter`. Para un proyecto Spring no Boot, asegúrate de tener:

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>6.1.3</version> <!-- Usa la versión más reciente -->
</dependency>
```

---

6. Ejemplo Completo con Spring Boot

Aquí tienes un ejemplo completo 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;
```

```java
import org.springframework.stereotype.Component;

@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("Ejecutándose cada 5 segundos: " + System.currentTimeMillis());
    }

    @Scheduled(cron = "0 */1 * * * *")
    public void runEveryMinute() {
        System.out.println("Ejecutándose cada minuto: " + System.currentTimeMillis());
    }
}
```

---

7. Personalizar el Programador (Opcional)

Por defecto, Spring usa un programador de un solo hilo. Si necesitas la ejecución concurrente de tareas, puedes configurar un `TaskScheduler` personalizado:

```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); // Número de hilos
        scheduler.setThreadNamePrefix("my-scheduler-");
        return scheduler;
    }
}
```

Esto permite que múltiples tareas se ejecuten en paralelo.

---

8. Probar tus Tareas Programadas

- Ejecuta tu aplicación y verifica los registros de la consola para asegurarte de que las tareas se estén ejecutando como se espera.
- Usa intervalos más cortos (por ejemplo, cada 2 segundos) durante las pruebas para ver los resultados rápidamente.