



Descargue el Informe de tendencias Kubernetes 2019 de DZone para ver el impacto futuro que tendrá Kubernetes.

[Descargar informe](#)

# Corriendo a tiempo con las tareas programadas de Spring

por Dan Newton · 06 de febrero, 18 · Zona Java · Tutorial

Get the fastest log management and analysis with Graylog open source or enterprise edition free up to 100GB per day.

Presented by Graylog

¿Necesita ejecutar un proceso todos los días exactamente a la misma hora como una alarma? Entonces las tareas programadas de Spring son para ti. Permitirle anotar un método con `@Scheduled` hace que se ejecute en el tiempo o intervalo específico que se denota dentro de él. En esta publicación, veremos cómo configurar un proyecto que pueda usar tareas programadas y cómo usar los diferentes métodos para definir cuándo se ejecutan.

Usaré Spring Boot para esta publicación, haciendo que las dependencias sean agradables y simples debido a que la programación está disponible para la `spring-boot-starter` dependencia que se incluirá en casi todos los proyectos Spring Boot de alguna manera. Esto le permite usar cualquiera de las otras dependencias de inicio, ya que se incorporarán `spring-boot-starter` y todas sus relaciones. Si desea incluir la dependencia exacta en sí misma, use `spring-context`.

Podrías usar `spring-boot-starter`.

```
1 < dependencia >
2   < groupId > org.springframework.boot </ groupId >
3   < artifactId > spring-boot-starter </ artifactId >
4   < versión > 2.0.0.RC1 </ versión >
5 </ dependencia >
```

O usar `spring-context` directamente.

```
1 < dependencia >
2   < groupId > org.springframework </ groupId >
3   < artifactId > spring-context </ artifactId >
4   < versión > 5.0.3.RELEASE </ version >
5 </ dependencia >
```

Crear una tarea programada es bastante sencillo. Agregue la `@Scheduled` anotación a cualquier método que desee ejecutar automáticamente e incluya `@EnableScheduling` en un archivo de configuración.

Entonces, por ejemplo, podrías tener algo como:

```
1 @Component
2 público de clase eventCreator {
3
4 }
```

```

3
4     Logger final estático privado LOG = LoggerFactory . getLogger ( EventCreator . clase );
5
6     evento privado final EventRepository eventRepository ;
7
8     pública eventCreator ( última EventRepository eventRepository ) {
9         esta . eventRepository = eventRepository ;
10    }
11
12    @Scheduled ( fixedRate = 1000 )
13    public void create () {
14        final LocalDateTime start = LocalDateTime . ahora ();
15        eventRepository . guardar (
16            nuevo Evento ( nueva EventKey ( "Un tipo de evento" , inicio , UUID . randomUUID () ), Ma
17            REGISTRO . debug ( "Evento creado!" );
18    }
19 }

```

Aquí hay bastante código que no tiene importancia para ejecutar una tarea programada. Como dije hace un minuto, necesitamos usar `@Scheduled` un método, y comenzará a ejecutarse automáticamente. Entonces, en el ejemplo anterior, el `create` método comenzará a ejecutarse cada 1000 ms (1 segundo) como lo indica la `fixedRate` propiedad de la anotación. Si quisiéramos cambiar la frecuencia con la que se ejecuta, podríamos aumentar o disminuir el `fixedRate` tiempo, o podríamos considerar usar los diferentes métodos de programación disponibles para nosotros.

Entonces, probablemente quieras saber cuáles son estas otras formas, ¿verdad? Bueno, aquí están (incluiré `fixedRate` aquí también):

- `fixedRate` ejecuta el método con un período fijo de milisegundos entre invocaciones.
- `fixedRateString` es lo mismo `fixedRate` pero con un valor de cadena en su lugar.
- `fixedDelay` ejecuta el método con un período fijo de milisegundos entre el final de una invocación y el inicio de la siguiente.
- `fixedDelayString` es lo mismo `fixedDelay` pero con un valor de cadena en su lugar.
- `cron` usa expresiones parecidas a cron para determinar cuándo ejecutar el método (lo veremos más en profundidad más adelante).

Hay algunas otras propiedades de utilidad disponibles para la `@Scheduled` anotación.

- `zone` indica la zona horaria en la que se resolverá la expresión cron. Si no se incluye una zona horaria, utilizará la zona horaria predeterminada del servidor. Entonces, si necesita que se ejecute para una zona horaria específica, digamos Hong Kong, podría usar `zone = "GMT+8:00"`.
- `initialDelay` es el número de milisegundos para retrasar la primera ejecución de una tarea programada. Requiere que se use una de las propiedades de velocidad fija o retraso fijo.
- `initialDelayString` es lo mismo `initialDelay` pero con un valor de cadena en su lugar.

A continuación se pueden encontrar algunos ejemplos del uso de tasas fijas y demoras:

Igual que antes: se ejecuta cada 1 segundo:

```

1 @Scheduled ( fixedRate = 1000 )

```

Lo mismo que arriba:

Se muestra que se ejecuta:

```
1 @Scheduled ( fixedRateString = "1000" )
```

Se ejecuta 1 segundo después de que finalice la invocación anterior:

```
1 @Scheduled ( fixedDelay = 1000 )
```

Se ejecuta cada segundo pero espera 5 segundos antes de ejecutarse por primera vez:

```
1 @Scheduled ( fixedRate = 1000 , initialDelay = 5000 )
```

Ahora en mirar la `cron` propiedad, que da mucho más control sobre la programación de una tarea. Nos permite definir los segundos, minutos y horas en que se ejecuta la tarea, pero puede ir más allá y especificar incluso los años en que se ejecutará una tarea.

A continuación se muestra un desglose de los componentes que crean una expresión cron.

- Seconds pueden tener valores 0-59 o los caracteres especiales , - \* / .
- Minutes pueden tener valores 0-59 o los caracteres especiales , - \* / .
- Hours pueden tener valores 0-23 o los caracteres especiales , - \* / .
- Day of month pueden tener valores 1-31 o los caracteres especiales , - \* ? / L W C .
- Month puede tener valores 1-12 , JAN-DEC o los caracteres especiales , - \* / .
- Day of week puede tener valores 1-7 , SUN-SAT o los caracteres especiales , - \* ? / L C # .
- Year puede estar vacío, tener valores 1970-2099 o los caracteres especiales , - \* / .

Solo para mayor claridad, he combinado el desglose en una expresión que consiste en las etiquetas de campo.

```
1 @Scheduled ( cron = "[Segundos] [Minutos] [Horas] [Día del mes] [Mes] [Día de la semana] [Año]" )
```

No incluya los corchetes en sus expresiones (los usé para aclarar la expresión).

Antes de poder seguir adelante, debemos analizar qué significan los caracteres especiales.

- \* representa todos los valores Entonces, si se usa en el segundo campo, significa cada segundo. Si se usa en el campo de día, significa ejecutar todos los días.
- ? no representa ningún valor específico y se puede usar en el campo del día del mes o del día de la semana, donde el uso de uno invalida al otro. Si especificamos que se active el día 15 de un mes, ? se utilizará a en el Day of week campo.
- - representa un rango inclusivo de valores. Por ejemplo, 1-3 en el campo de horas significa las horas 1, 2 y 3.
- , representa valores adicionales Por ejemplo, poner MON, WED, SUN en el campo del día de la semana significa lunes, miércoles y domingo.
- / representa incrementos. Por ejemplo, 0/15 en el campo de segundos se dispara cada 15 segundos a partir de 0 (0, 15, 30 y 45).
- L representa el último día de la semana o mes. Recuerde que el sábado es el final de la semana en este contexto, por lo que usarlo L en el campo del día de la semana se activará un sábado. Esto se puede usar junto con un número en el campo del día del mes, como 6L para representar el último viernes del mes o una expresión como L-3 denotar el tercero desde el último día del mes. Si especificamos un valor en el campo del día de la semana,

debemos usarlo ? en el campo del día del mes y viceversa.

- w representa el día de la semana más cercano del mes. Por ejemplo, 15w se activará el día 15 del mes si es un día de la semana. De lo contrario, se ejecutará el día de la semana más cercano. Este valor no se puede usar en una lista de valores de día.
- # especifica tanto el día de la semana como la semana en que la tarea debería desencadenarse. Por ejemplo, 5#2 significa el segundo jueves del mes. Si el día y la semana que especificó se desbordan en el próximo mes, no se activará.

Puede encontrar un recurso útil con explicaciones un poco más largas aquí , que me ayudó a escribir esta publicación.

Veamos algunos ejemplos:

Incendios a las 12 p. M. Todos los días:

```
1 @Scheduled ( cron = "0 0 12 * *?" )
```

Incendios a las 10:15 a.m.todos los días del año 2005:

```
1 @Scheduled ( cron = "0 15 10 * *? 2005" )
```

Dispara cada 20 segundos:

```
1 @Scheduled ( cron = "0/20 * * * *?" )
```

Para ver algunos ejemplos más, vea el enlace que mencioné anteriormente, que se muestra nuevamente aquí .

Afortunadamente, si te quedas atascado al escribir una expresión cron simple, deberías poder buscar en Google el escenario que necesitas, ya que probablemente alguien ya ha hecho la misma pregunta sobre Stack Overflow.

Para vincular algunas de las lecciones anteriores en un pequeño ejemplo de código, consulte el siguiente código:

```
1 @Component
2 public class AverageMonitor {
3
4     Logger final estático privado LOG = LoggerFactory . getLogger ( AverageMonitor . class );
5     evento privado final EventRepository eventRepository ;
6     Private Final AverageRepository averageRepository ;
7
8     pública AverageMonitor (
9         final EventRepository eventRepository , final AverageRepository averageRepository ) {
10         esta . eventRepository = eventRepository ;
11         esta . averageRepository = averageRepository ;
12     }
13
14     @Scheduled ( cron = "0/20 * * * *?" )
15     público nulo publicar () {
16         promedio doble final =
17             eventRepository . getAverageValueGreaterThanStartTime (
18                 "Un tipo de evento" , LocalDateTime . ahora () . minusSeconds ( 20 ));
19         mediaRepository . guardar (
20             new Average ( new AverageKey ( "Un tipo de evento" , LocalDateTime . now () ), average ));
21         REGISTRO . info ( "El valor promedio es {}" , promedio );
22     }
23 }
```

```
22     }  
23 }
```

Aquí, tenemos una clase que consulta a Cassandra cada 20 segundos por el valor promedio de los eventos en el mismo período de tiempo. Nuevamente, la mayor parte del código aquí es ruido de la `@Scheduled` anotación, pero puede ser útil verlo en la naturaleza. Además, si ha sido observador, para este caso de uso de ejecución cada 20 segundos, usar las `fixedRate` y posiblemente las `fixedDelay` propiedades en lugar de `cron` sería adecuado aquí ya que ejecutamos la tarea con tanta frecuencia:

```
1 @Scheduled ( fixedRate = 20000 )
```

Es el `fixedRate` equivalente de la expresión `cron` utilizada anteriormente.

El requisito final, al que aludí anteriormente, es agregar la `@EnableScheduling` anotación a una clase de configuración:

```
1 @SpringBootApplication  
2 @EnableScheduling  
3 Solicitud de clase pública {  
4  
5     public static void main ( args finales de cadena []) {  
6         SpringApplication . ejecutar ( aplicación . clase );  
7     }  
8 }
```

Siendo que esta es una pequeña aplicación Spring Boot, he adjuntado la `@EnableScheduling` anotación a la `@SpringBootApplication` clase principal .

En conclusión, podemos programar tareas para que se activen utilizando la `@Scheduled` anotación junto con una tasa de milisegundos entre ejecuciones o una expresión `cron` para tiempos más finos que no se pueden expresar con el primero. Para las tareas que deben ejecutarse con mucha frecuencia, bastará con las propiedades `fixedRate` o `fixedDelay` , pero una vez que el tiempo entre ejecuciones sea mayor, será más difícil determinar rápidamente el tiempo definido. Cuando esto ocurre, la `cron` propiedad debe usarse para una mejor claridad del tiempo programado.

La pequeña cantidad de código utilizada en esta publicación se puede encontrar en mi [GitHub](#) .

Si esta publicación le resultó útil y desea mantenerse al día con mis nuevos tutoriales mientras los escribo, síganme en Twitter en [@LankyDanDev](#) .

---

[NEW TREND REPORT] Scaling DevOps. Automation, culture, collaboration, and tools to scale DevC  
[Today](#)

Presented by DZone

---

## ¿Te gusta este artículo? Leer más de DZone



**Implementación de un bloqueo de programador**



**Cómo programar una tarea con un retraso fijo en Spring Boot [Video]**



**Cómo programar una tarea usando la expresión de Cron en Spring Boot**




**Tarjeta DZone gratis  
Java 13**

**[Video]**



Temas: JAVA, TAREAS PROGRAMADAS, PRIMAVERA, ARRANQUE DE PRIMAVERA, TUTORIAL

Publicado en DZone con permiso de Dan Newton , DZone MVB . [Vea el artículo original aquí.](#)   
Las opiniones expresadas por los contribuyentes de DZone son propias.