# Spring使用@Scheduled注解

Spring 提供了@Scheduled 注解，良好的解决了定时任务的需求，它的实现本质是基于 java 中的 ScheduledExecutorService 类的 schedule 方法。

@Scheduled 注解标注在方法上，它是 Spring 实现的一种计划任务，可以支持如下几种方式运行：

1. 固定时间频率运行方法。
2. 延迟指定的时间运行方法。
3. 按照 cron 表达式定义的时间方式运行方法。

### 使用@Scheduled 注解

1. 配置文件/配置类中，打开运行执行 Schedule 任务的开关，使用注解 @EnableScheduling。

@Configuration  
@ComponentScan("com.example")  
@EnableScheduling *//1.通过@EnableScheduling注解开启对计划任务的支持*public class TaskSchedulerConfig {  
  
}

1. 将某个类的方法标注@Scheduled。

@Slf4j  
@Service *//@Service注解为Service类并注册到spring容器中*public class ScheduledTaskService {  
private static final SimpleDateFormat ***dateFormat*** = new SimpleDateFormat("HH:mm:ss");  
@Scheduled(fixedRate = 1000,zone = "Asia/Shanghai")  
 public void fixedRateRun() {  
 ***log***.info("fixedRate：\t\t每隔一秒执行一次 " + ***dateFormat***.format(new Date()));  
 }  
@Scheduled(cron = "0 28 11 \* \* \*" )  
 public void cornTimeRun() {  
 ***log***.info("cron-Time：\t\t在指定时间执行 " + ***dateFormat***.format(new Date()));  
 }}

3. 启动 Spring 容器应用

 @Slf4j  
public class Main {  
 public static void main(String[] args) {  
  
 *//使用AnnotationConfigApplicationContext实现基于Java的配置类加载Spring的应用上下文* AnnotationConfigApplicationContext context =  
 new AnnotationConfigApplicationContext(TaskSchedulerConfig.class);  
   
 }  
}

@Schduled 默认是基于单线程执行, 所有的定时任务串行执行，这就可能导致运行时间久的任务，会影响到下一个运行周期的任务。如果需要基于多线程执行，则需要配置。

可以在配置文件中加入如下代码：

<task:annotation-driven scheduler="scheduler"/>

<task:scheduler id="scheduler" pool-size="5"/>

### 参数详解

### 1、cron

该参数接收一个cron表达式，cron表达式是一个字符串，字符串以5或6个空格隔开，分开共6或7个域，每一个域代表一个含义。

**cron表达式语法 ：[秒] [分] [小时] [日] [月] [周] [年]**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **说明** | **是否必填** | **允许填写的值** | **允许的通配符** |
| 1 | 秒 | 是 | 0-59 | , - \* / |
| 2 | 分 | 是 | 0-59 | , - \* / |
| 3 | 时 | 是 | 0-23 | , - \* / |
| 4 | 日 | 是 | 1-31 | , - \* / |
| 5 | 年 | 是 | 1-12 | , - \* ? / L W |
| 6 | 周 | 是 | 1-7 | , - \* ? / L # |
| 7 | 年 | 否 | 1970-2099 | , - \* / |

**通配符说明**

1. \*表示所有值。 例如:在分的字段上设置 \*,表示每一分钟都会触发。
2. ? 表示不指定值。使用的场景为不需要关心当前设置这个字段的值。例如:要在每月的10号触发一个操作，但不关心是周几，所以需要周位置的那个字段设置为”?” 具体设置为 0 0 0 10 \* ?
3. -表示区间。例如 在小时上设置 “10-12”,表示 10,11,12点都会触发。
4. , 表示指定多个值，例如在周字段上设置 “MON,WED,FRI” 表示周一，周三和周五触发
5. / 用于递增触发。如在秒上面设置”5/15” 表示从5秒开始，每增15秒触发(5,20,35,50)。 在月字段上设置’1/3’所示每月1号开始，每隔三天触发一次。
6. L 表示最后的意思。在日字段设置上，表示当月的最后一天(依据当前月份，如果是二月还会依据是否是润年[leap]), 在周字段上表示星期六，相当于”7”或”SAT”。如果在”L”前加上数字，则表示该数据的最后一个。例如在周字段上设置”6L”这样的格式,则表示“本月最后一个星期五”
7. W 表示离指定日期的最近那个工作日(周一至周五). 例如在日字段上置”15W”，表示离每月15号最近的那个工作日触发。如果15号正好是周六，则找最近的周五(14号)触发, 如果15号是周未，则找最近的下周一(16号)触发.如果15号正好在工作日(周一至周五)，则就在该天触发。如果指定格式为 “1W”,它则表示每月1号往后最近的工作日触发。如果1号正是周六，则将在3号下周一触发。(注，”W”前只能设置具体的数字,不允许区间”-“)。
8. #序号(表示每月的第几个周几)，例如在周字段上设置”6#3”表示在每月的第三个周六.注意如果指定”#5”,正好第五周没有周六，则不会触发该配置(用在母亲节和父亲节再合适不过了) ；小提示：’L’和 ‘W’可以一组合使用。如果在日字段上设置”LW”,则表示在本月的最后一个工作日触发；周字段的设置，若使用英文字母是不区分大小写的，即MON与mon相同。

**示例**

每隔5秒执行一次：\*/5 \* \* \* \* ?

每隔1分钟执行一次：0 \*/1 \* \* \* ?

每天23点执行一次：0 0 23 \* \* ?

每天凌晨1点执行一次：0 0 1 \* \* ?

每月1号凌晨1点执行一次：0 0 1 1 \* ?

每月最后一天23点执行一次：0 0 23 L \* ?

每周星期天凌晨1点实行一次：0 0 1 ? \* L

在26分、29分、33分执行一次：0 26,29,33 \* \* \* ?

每天的0点、13点、18点、21点都执行一次：0 0 0,13,18,21 \* \* ?

*/\*\*  
 \* 每天11点28分执行  
 \*/*@Scheduled(cron = "0 28 11 \* \* \*" )  
public void cornTimeRun() {  
 ***log***.info("cron-Time：\t在指定时间执行 " + ***dateFormat***.format(new Date()));  
}

*/\*\*  
 \* 每隔5秒执行一次，任务开启后第一次不会执行，会5秒后首次执行  
 \*/*@Scheduled(cron = "\*/5 \* \* \* \* \*" )  
public void cornSpanRun() {  
 ***log***.info("cron-Span：\t每隔五秒执行一次 " + ***dateFormat***.format(new Date()));  
}

### 2. zone

时区，接收一个java.util.TimeZone#ID。cron表达式会基于该时区解析。默认是一个空字符串，即取服务器所在地的时区。比如我们一般使用的时区Asia/Shanghai。该字段我们一般留空。

@Scheduled(fixedRate = 1000,zone = "Asia/Shanghai")  
public void fixedRateRun() {  
 ***log***.info("fixedRate：\t每隔一秒执行一次 " + ***dateFormat***.format(new Date()));  
}

### 3. fixedDelay

上一次执行完毕时间点之后多长时间再执行。如：

*/\*\*  
 \*使用fixedDelay属性每隔固定时间（单位：毫秒）异步执行  
 \* 每隔5秒执行一次,上一次执行完毕时间点之后多长时间再执行  
 \*/*@Scheduled(fixedDelay = 5000)  
public void fixedDelayRun() {  
 ***log***.info("fixedDelay：\t每隔五秒执行一次 " + ***dateFormat***.format(new Date()));  
}

### 

### 4. fixedDelayString

与 3. fixedDelay 意思相同，只是使用字符串的形式。唯一不同的是支持占位符。如：

@Scheduled(fixedDelayString = "5000}")//上一次执行完毕时间点之后5秒再执行

占位符的使用(配置文件中有配置：time.fixedDelay=5000)：

*/\*\*  
 \* 与fixedDelay相同，只是使用字符串的形式。并支持占位符。如：  
 \* @Scheduled(fixedDelayString = "5000")  
 \* 占位符的使用(配置文件中有配置：time.fixedDelay=5000)：  
 \*/*@Scheduled(fixedDelayString = "${time.fixedDelay}")  
void FixedDelayStringRun() {  
 ***log***.info("FixedDelayString：\t读取配置文件执行" + System.currentTimeMillis());  
}

### 

### 5. fixedRate

上一次开始执行时间点之后多长时间再执行。如：

*/\*\*  
 \*通过@Scheduled声明该方法是计划任务，使用fixedRate属性每隔固定时间（单位：毫秒）异步执行  
 \* 每隔1秒执行一次，上一次开始执行时间点之后5秒再执行  
 \*/*@Scheduled(fixedRate = 1000)  
public void fixedRateRun() {  
 ***log***.info("fixedRate：\t每隔一秒执行一次 " + ***dateFormat***.format(new Date()));  
}

### 6. fixedRateString

与 fixedRate 意思相同，只是使用字符串的形式。唯一不同的是支持占位符。

*/\*\*  
 \* 与fixedRate相同，只是使用字符串的形式。并支持占位符。如：  
 \* @Scheduled(fixedDelayString = "1000")  
 \* 占位符的使用(配置文件中有配置：time.fixedRate=1000)：  
 \*/*@Scheduled(fixedRateString = "1000")  
public void fixedRateStringRun() {  
 ***log***.info("fixedRateString：\t每隔一秒执行一次 " + ***dateFormat***.format(new Date()));  
}

### 7. initialDelay

第一次延迟多长时间后再执行。如：

*/\*\*  
 \* 使用initialDelay属性第一次延迟多长时间后再执行  
 \* 第一次延迟10秒后执行，之后按fixedRate的规则每5秒执行一次  
 \*/*@Scheduled(initialDelay=10000, fixedRate=5000)  
void initialDelayRun() {  
 ***log***.info("initialDelay：\t延迟10秒执行" + System.currentTimeMillis());  
}

### 8. initialDelayString

与 initialDelayString 意思相同，只是使用字符串的形式。唯一不同的是支持占位符。

*/\*\*  
 \* 与initialDelay相同，只是使用字符串的形式。并支持占位符。如：  
 \* @Scheduled(fixedDelayString = "10000",fixedRate=5000)  
 \* 占位符的使用(配置文件中有配置：time.initialDelay=10000)：  
 \*/*@Scheduled(initialDelayString="${time.initialDelay}", fixedRate=5000)  
void initialDelayStringRun() {  
 ***log***.info("initialDelayStr：\t延迟10秒执行" + System.currentTimeMillis());  
}