java并发编程笔记--任务调度

|  |  |
| --- | --- |
|  | 🢂 内容概览 |
|  | Why：此文档用来做什么？它存在的意义是什么？为解决什么问题？   |  | | --- | |  |   What：当前包含了那些内容？   |  | | --- | |  |   How：此文档应如何参考？   |  | | --- | |  |   Who：此文档适用于那些人员阅读参考？   |  | | --- | |  |   Summary：摘要   |  | | --- | | TaskExecutor接口  TaskScheduler接口  Trigger&TriggerContext接口  @EnableAsync & @EnableScheduling注解  @Scheduled & @Async注解  task命名空间  JobDetailFacotryBean & MethodInvokingJobDetailFacotryBean  SchedulerFactoryBean  CronTriggerFactoryBean & SimpleTriggerFactoryBean  AsyncUncaughtExceptionHandler |   Reference：参考文献   |  | | --- | | [CommonJ简介](http://developer.zdnet.com.cn/2007/1003/534296.shtml)  [一张图让你秒懂Spring @Scheduled定时任务的fixedRate,fixedDelay,cron执行差异](http://blog.csdn.net/applebomb/article/details/52400154)  [深入学习spring：注解](http://blog.csdn.net/yupeikang/article/details/50294297) | |

目录

[1 简介 4](#_Toc462847642)

[2 TaskExecutor 4](#_Toc462847643)

[2.1 TaskExecutor实现类 4](#_Toc462847644)

[2.2 示例：使用TaskExecutor 5](#_Toc462847645)

[3 TaskScheduler 5](#_Toc462847646)

[3.1 fixed-delay & fixed-rate & cron 6](#_Toc462847647)

[3.2 Trigger接口 7](#_Toc462847648)

[3.3 Trigger的实现类 7](#_Toc462847649)

[3.4 TaskScheduler实现 7](#_Toc462847650)

[4 基于注解方式配置 7](#_Toc462847651)

[4.1 设置@Scheduled和@Async注解支持； 7](#_Toc462847652)

[4.2 @Scheduled注解 8](#_Toc462847653)

[4.3 @Async注解 8](#_Toc462847654)

[5 基于xml方式配置 10](#_Toc462847655)

[5.1 线程池的执行逻辑 10](#_Toc462847656)

[6 使用Quartz 11](#_Toc462847657)

[6.1 创建Job 12](#_Toc462847658)

[6.1.1 JobDetailFacotryBean 12](#_Toc462847659)

[6.1.2 MethodInvokingJobDetailFactoryBean 12](#_Toc462847660)

[6.2 创建Trigger 13](#_Toc462847661)

[7 问题思考 14](#_Toc462847662)

# 简介

1. spring framework通过TaskExecutor和TaskScheduler两个接口提供了异步执行和任务调度的抽象，能够支持线程池，以及commonJ等；
2. spring framework同时提供了用于支持Timer，Quartz的类；这些类都是通过FactoryBean建立，可以附加TimerTask或者Trigger属性；能够指定Timer或者Quartz执行指定的bean方法；

# TaskExecutor

1. TaskExecutor是spring用来提供任务执行的接口，接口的底部可以是一个线程池，单个线程甚至是同步执行；
2. TaskExecutor和java.util.concurrent.Executor的意义相同，之所以单独定义，主要是为了屏蔽不同版本JDK对线程池的支持；
3. TaskExecutor用于为spring其它组件提供线程池支持，比如：ApplicationEventMulticaster ，JMS’s AbstractMessageListenerContainer 以及Quartz 的集成，也可以在业务中使用该接口作为线程池入口；

## TaskExecutor实现类

1. **SimpleAsyncTaskExecutor**：并非真正的线程池实现，不能够重用线程，只是为每个调用都新建一个线程，可以限定任务数，当调用超过任务数限定时，将会被阻塞，直到有下一个slot空闲时，才唤醒；
2. **SyncTaskExecutor**：直接在调用线程中执行任务，适用于不需要多线程的场景，比如：测试用例执行的场景；
3. **ConcurrentTaskExecutor**：作为java.util.concurrent.Executor接口的实现类。适用于适配不同种类的java.util.concurrent.Executor；
4. **SimpleThreadPoolTaskExecutor**：作为quartz的SimpleThreadPool的子类，能够监听spring的生命周期回调；即适用于quartz，也适用于非quartz环境；
5. **ThreadPoolTaskExecutor**：作为TaskExecutor最通用的实现，用于适配java.util.concurrent.ThreadPoolExecutor；如果需要适配java.util.concurrent.Executor接口的其它实现，则推荐使用ConcurrentTaskExecutor；
6. **WorkManagerTaskExecutor**：使用CommonJ的WorkManager作为底层实现，用于在spring context中建立和使用WorkManager；

## 示例：使用TaskExecutor

1. 定义要执行的任务：

|  |
| --- |
|  |

1. 配置线程池：

|  |
| --- |
|  |

# TaskScheduler

TaskScheduler接口用于控制任务按照指定的规律执行，比如：在到达某一时间点才执行；接口定义如下：

|  |
| --- |
|  |

## fixed-delay & fixed-rate & cron

|  |
| --- |
|  |

1. cron方式：如果任务执行时间超过执行周期，则delay到下一个周期执行下一次任务；
2. fixed-delay方式：如果任务执行时间超过执行周期，则delay到下一个周期执行下一次任务；
3. fixed-rate方式：如果任务执行时间超过执行周期，则缓存任务，等待当前任务执行完成后立刻执行下一次任务；

## Trigger接口

作为JSR-236的实现，用于支持复杂的执行规则的定制，比如：需要依赖上一次执行的情况来确定当前执行规则；如果需要获取之前的执行内容，则可以在TriggerContext中获取；

|  |
| --- |
|  |

如下是TriggerContext的接口定义：

|  |
| --- |
|  |

## Trigger的实现类

1. CronTrigger：支持cron表达式的触发器；

|  |
| --- |
|  |

1. PeriodicTrigger：支持延迟一定时间的触发器；

## TaskScheduler实现

1. TimerManagerTaskScheduler：封装CommonJ的TimerManager，通过JNDI-lookup配置；
2. ThreadPoolTaskScheduler：封装ScheduledExecutorService实例，同时实现了TaskExecutor接口，所以即可以用作异步调用，又可以用作任务调度；

# 基于注解方式配置

## 设置@Scheduled和@Async注解支持；

1. 基于Java的配置方式

|  |
| --- |
|  |

1. 基于xml的配置方式：

|  |
| --- |
|  |

## @Scheduled注解

@Scheduled注解用于指定被spring调度的方法。注解添加到方法名上，可以附带触发规则参数；

1. 每5秒调用1次方法，以fixedDelay方式执行；

|  |
| --- |
|  |

1. 每5秒调用1次方法，以fixedRate方式执行；

|  |
| --- |
|  |

1. 第一次执行延迟1秒，以后每5秒调用1次方法，以fixedRate方式执行；

|  |
| --- |
|  |

1. 每周五，每隔5秒执行一次，以cron方式执行，可以通过zone属性指定时区；

|  |
| --- |
|  |

注：

1. 被执行的方法必须返回void，并且没有参数；如果需要依赖IOC容器中的其它对象，则需要通过依赖注入的方式；
2. 确保没有初始化多个被@Scheduled注解的类的实例，除非想要在每个实例都调度这些方法；
3. 确保被@Scheduled注解的类没有被@Configurable注解，否则方法将会被调度两次，一次是通过容器调用，一次是通过@Configurable切面调用；

## @Async注解

@Async标注方法异步执行，调用方法会立刻返回；而调用内容将会被提交到TaskExecutor执行；调用方法需要返回void或者Future类型；

1. 无参数的异步方法；

|  |
| --- |
|  |

1. 有参数的异步方法，能够按照调用普通方法的方式调用；

|  |
| --- |
|  |

1. 有返回值的异步方法，返回值需要通过Future包装；

|  |
| --- |
|  |

1. 使用指定的Executor调用异步方法而非普通方法调用；

|  |
| --- |
|  |

注意：

1. @Async不能够和@PostConstruct注解同时使用，如果要在@PostConstruct中异步调用方法，需要将@Async注解的方法放入到单独的Spring Bean中；示例如下：

|  |
| --- |
|  |

1. 当@Async方法返回Future时，抛出的异常可以通过get()方法捕获；当返回void时，将不容易捕获，AsyncUncaughtExceptionHandler接口用于处理这些异常，出现异常时默认将会打印日志，可以通过定义实现类，更改对异常处理操作；自定义的实现类通过AsyncConfigurer或者task:annotation-driven配置；

|  |
| --- |
|  |

# 基于xml方式配置

task命名空间（xmlns:task="http://www.springframework.org/schema/task），用于简化任务调度相关的xml配置；

1. 创建TaskScheduler，将创建ThreadPoolTaskScheduler实例，id指定的名称将作为线程池中线程名字的前缀，如果不指定pool-size，将默认为1；task:scheduler仅支持pool-size配置；

|  |
| --- |
|  |

1. 创建TaskExecutor，将创建ThreadPoolTaskExecutor实例，id指定的名称将作为线程池中线程名字的前缀，如果不指定pool-size，将默认为1；相比task:scheduler，task:executor可以支持更多的参数配置，比如：pool-size指定范围；

|  |
| --- |
|  |

1. 创建调度任务：能够指定调度调度IOC容器中指定实例的指定方法；

|  |
| --- |
|  |

1. 创建调度任务：使用cron方式触发任务；

|  |
| --- |
|  |

## 线程池的执行逻辑

|  |
| --- |
|  |

# 使用Quartz

1. Quartz将一个任务分解为：Trigger，Job，JobDetail三部分，由Scheduler来调度任务；
2. Trigger定义任务的执行策略，比如：什么时间执行，执行几次等；
3. Job定义任务执行的内容，类似Runnable接口；
4. JobDetail用来保存任务的执行信息，包括执行任务所需的参数；

spring提供了一些FacotryBean类，用于简化quartz的使用；

## 创建Job

### JobDetailFacotryBean

用于定义jobdetail实例，jobdetail中保存的参数，默认以map方式保存，通过JobExecutionContext访问；如果QuartzJobBean中包含指定参数的属性名，则可以注入参数值到QuartzJobBean中；

|  |
| --- |
|  |

|  |
| --- |
|  |

### MethodInvokingJobDetailFactoryBean

MethodInvokingJobDetailFactoryBean可以指定需要调度的方法；

|  |
| --- |
|  |

|  |
| --- |
|  |

MethodInvokingJobDetailFactoryBean可以指定多个trigger是否可以并发执行，默认可以并发；因为Job是无状态的，如果存在多个trigger调度同一个job，则有可能导致一个trigger没有执行完，另一个trigger就开始执行了，可以通过concurrent参数关闭并发执行，保证trigger不能并发执行；

|  |
| --- |
|  |

## 创建Trigger

1. spring提供了两种创建Trigger的Factory：CronTriggerFactoryBean和SimpleTriggerFactoryBean，分别用来创建CronTrigger和SimpleTrigger；
2. spring提供了SchedulerFactoryBean用来创建Scheduler，用来调度任务；

定义trigger示例：

|  |
| --- |
|  |

定义Scheduler示例：

|  |
| --- |
|  |

# 问题思考

1）@Configurable注解的作用是什么？

2）为什么固定大小的线程池，要使用无界的缓冲队列？