# spring schedule定时任务

**文章目录**

spring schedule定时任务

一、如何使用定时任务

1、启动类使用@EnableScheduling注解开启定时任务

2、方法使用@Scheduled注解，或者实现SchedulingConfigurer接口，添加定时任务

二、配置定时任务多线程非阻塞运行

1、阻塞原因

2、如何解决，实现SchedulingConfigurer接口，设置任务调度器实现类

三、源码解析

1、@EnableScheduling注解启用定时任务

2、扫描@Scheduled、@Schedules注解

3、扫描SchedulingConfigurer实现类

4、添加定时任务到线程池

## **一、如何使用定时任务**

### **1、启动类使用@**[EnableScheduling](https://so.csdn.net/so/search?q=EnableScheduling&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_34480904/article/details/_blank)**注解开启定时任务**

@SpringBootApplication

@EnableScheduling

public class ScheduledTest {

public static void main(String[] args) {

SpringApplication.run(ScheduledTest.class);

}

}

### **2、方法使用@[Scheduled](https://so.csdn.net/so/search?q=Scheduled&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_34480904/article/details/_blank)注解，或者实现SchedulingConfigurer接口，添加定时任务**

@Scheduled(cron = "0/2 \* \* \* \* ? ")

public void index1() {

log.info("定时任务1，2秒执行一次，time：" + DateTime.now() + " 线程：" + Thread.currentThread().getName());

}

@Configuration

@Component

@Slf4j

public class TestTask implements SchedulingConfigurer {

@Override

public void configureTasks(ScheduledTaskRegistrar taskRegistrar) {

taskRegistrar.addFixedDelayTask(this::index3, 1000);

}

public void index2() {

log.info("定时任务2，1秒执行一次，time：" + DateTime.now() + " 线程：" + Thread.currentThread().getName());

}

}

## **二、配置定时任务多线程非阻塞运行**

### **1、阻塞原因**

默认情况，定时任务使用的是单例线程执行器Executors.newSingleThreadScheduledExecutor()，所以当一个定时任务阻塞是，所有定时任务都不会执行：

public class ScheduledTaskRegistrar implements ScheduledTaskHolder, InitializingBean, DisposableBean {

@SuppressWarnings("deprecation")

protected void scheduleTasks() {

if (this.taskScheduler == null) {

this.localExecutor = Executors.newSingleThreadScheduledExecutor();

this.taskScheduler = new ConcurrentTaskScheduler(this.localExecutor);

}

if (this.triggerTasks != null) {

for (TriggerTask task : this.triggerTasks) {

addScheduledTask(scheduleTriggerTask(task));

}

}

if (this.cronTasks != null) {

for (CronTask task : this.cronTasks) {

addScheduledTask(scheduleCronTask(task));

}

}

if (this.fixedRateTasks != null) {

for (IntervalTask task : this.fixedRateTasks) {

addScheduledTask(scheduleFixedRateTask(task));

}

}

if (this.fixedDelayTasks != null) {

for (IntervalTask task : this.fixedDelayTasks) {

addScheduledTask(scheduleFixedDelayTask(task));

}

}

}

}

### **2、如何解决，实现SchedulingConfigurer接口，设置任务调度器实现类**

使用线程池执行定时任务，ThreadPoolExecutor.AbortPolicy:丢弃任务并抛出RejectedExecutionException异常。

@Configuration

@Component

@Slf4j

public class TestTask implements SchedulingConfigurer {

@Override

public void configureTasks(ScheduledTaskRegistrar taskRegistrar) {

ThreadFactory nameThreadFactory = new ThreadFactoryBuilder().setNameFormat("scheduled-%d").build();

taskRegistrar.setScheduler(new ScheduledThreadPoolExecutor(5,

nameThreadFactory,

new ThreadPoolExecutor.AbortPolicy()));

}

}

## **三、源码解析**

### **1、@EnableScheduling注解启用定时任务**

打开注解，发现这里只是在引用SchedulingConfiguration.class配置

@Target(ElementType.TYPE)

@Retention(RetentionPolicy.RUNTIME)

@Import(SchedulingConfiguration.class)

@Documented

public @interface EnableScheduling {

}

打开配置类，发现是在实例化ScheduledAnnotationBeanPostProcessor类

@Configuration(proxyBeanMethods = false)

@Role(BeanDefinition.ROLE\_INFRASTRUCTURE)

public class SchedulingConfiguration {

@Bean(name = TaskManagementConfigUtils.SCHEDULED\_ANNOTATION\_PROCESSOR\_BEAN\_NAME)

@Role(BeanDefinition.ROLE\_INFRASTRUCTURE)

public ScheduledAnnotationBeanPostProcessor scheduledAnnotationProcessor() {

return new ScheduledAnnotationBeanPostProcessor();

}

}

注解Bean后置处理器初始化方法是在实例化ScheduledTaskRegistrar类

public class ScheduledAnnotationBeanPostProcessor

implements ScheduledTaskHolder, MergedBeanDefinitionPostProcessor, DestructionAwareBeanPostProcessor,

Ordered, EmbeddedValueResolverAware, BeanNameAware, BeanFactoryAware, ApplicationContextAware,

SmartInitializingSingleton, ApplicationListener<ContextRefreshedEvent>, DisposableBean {

/\*\*

\* The default name of the {@link TaskScheduler} bean to pick up: {@value}.

\* <p>Note that the initial lookup happens by type; this is just the fallback

\* in case of multiple scheduler beans found in the context.

\* @since 4.2

\*/

public static final String DEFAULT\_TASK\_SCHEDULER\_BEAN\_NAME = "taskScheduler";

protected final Log logger = LogFactory.getLog(getClass());

private final ScheduledTaskRegistrar registrar;

@Nullable

private Object scheduler;

private final Set<Class<?>> nonAnnotatedClasses = Collections.newSetFromMap(new ConcurrentHashMap<>(64));

private final Map<Object, Set<ScheduledTask>> scheduledTasks = new IdentityHashMap<>(16);

/\*\*

\* Create a default {@code ScheduledAnnotationBeanPostProcessor}.

\*/

public ScheduledAnnotationBeanPostProcessor() {

this.registrar = new ScheduledTaskRegistrar();

}

}

### **2、扫描@Scheduled、@Schedules注解**

ScheduledAnnotationBeanPostProcessor实现了BeanPostProcessor接口。调用postProcessAfterInitialization后置处理器扫描注解，全部转换为Scheduled后，调用processScheduled方法

@Override

public Object postProcessAfterInitialization(Object bean, String beanName) {

if (bean instanceof AopInfrastructureBean || bean instanceof TaskScheduler ||

bean instanceof ScheduledExecutorService) {

// Ignore AOP infrastructure such as scoped proxies.

return bean;

}

Class<?> targetClass = AopProxyUtils.ultimateTargetClass(bean);

if (!this.nonAnnotatedClasses.contains(targetClass) &&

AnnotationUtils.isCandidateClass(targetClass, Arrays.asList(Scheduled.class, Schedules.class))) {

Map<Method, Set<Scheduled>> annotatedMethods = MethodIntrospector.selectMethods(targetClass,

(MethodIntrospector.MetadataLookup<Set<Scheduled>>) method -> {

Set<Scheduled> scheduledAnnotations = AnnotatedElementUtils.getMergedRepeatableAnnotations(

method, Scheduled.class, Schedules.class);

return (!scheduledAnnotations.isEmpty() ? scheduledAnnotations : null);

});

if (annotatedMethods.isEmpty()) {

this.nonAnnotatedClasses.add(targetClass);

if (logger.isTraceEnabled()) {

logger.trace("No @Scheduled annotations found on bean class: " + targetClass);

}

}

else {

// Non-empty set of methods

annotatedMethods.forEach((method, scheduledAnnotations) ->

scheduledAnnotations.forEach(scheduled -> processScheduled(scheduled, method, bean)));

if (logger.isTraceEnabled()) {

logger.trace(annotatedMethods.size() + " @Scheduled methods processed on bean '" + beanName +

"': " + annotatedMethods);

}

}

}

return bean;

}

1、先创建执行runnable

2、获取延迟执行时间

3、获取cron表达式，创建CronTask，registrar中添加任务

4、获取固定延迟，创建FixedDelayTask，registrar中添加任务

5、获取固定执行间隔，创建FixedRateTask，registrar中添加任务

6、把所有任务都添加到scheduledTasks

protected void processScheduled(Scheduled scheduled, Method method, Object bean) {

try {

Runnable runnable = createRunnable(bean, method);

boolean processedSchedule = false;

String errorMessage =

"Exactly one of the 'cron', 'fixedDelay(String)', or 'fixedRate(String)' attributes is required";

Set<ScheduledTask> tasks = new LinkedHashSet<>(4);

// Determine initial delay

long initialDelay = scheduled.initialDelay();

String initialDelayString = scheduled.initialDelayString();

if (StringUtils.hasText(initialDelayString)) {

Assert.isTrue(initialDelay < 0, "Specify 'initialDelay' or 'initialDelayString', not both");

if (this.embeddedValueResolver != null) {

initialDelayString = this.embeddedValueResolver.resolveStringValue(initialDelayString);

}

if (StringUtils.hasLength(initialDelayString)) {

try {

initialDelay = parseDelayAsLong(initialDelayString);

}

catch (RuntimeException ex) {

throw new IllegalArgumentException(

"Invalid initialDelayString value \"" + initialDelayString + "\" - cannot parse into long");

}

}

}

// Check cron expression

String cron = scheduled.cron();

if (StringUtils.hasText(cron)) {

String zone = scheduled.zone();

if (this.embeddedValueResolver != null) {

cron = this.embeddedValueResolver.resolveStringValue(cron);

zone = this.embeddedValueResolver.resolveStringValue(zone);

}

if (StringUtils.hasLength(cron)) {

Assert.isTrue(initialDelay == -1, "'initialDelay' not supported for cron triggers");

processedSchedule = true;

if (!Scheduled.CRON\_DISABLED.equals(cron)) {

TimeZone timeZone;

if (StringUtils.hasText(zone)) {

timeZone = StringUtils.parseTimeZoneString(zone);

}

else {

timeZone = TimeZone.getDefault();

}

tasks.add(this.registrar.scheduleCronTask(new CronTask(runnable, new CronTrigger(cron, timeZone))));

}

}

}

// At this point we don't need to differentiate between initial delay set or not anymore

if (initialDelay < 0) {

initialDelay = 0;

}

// Check fixed delay

long fixedDelay = scheduled.fixedDelay();

if (fixedDelay >= 0) {

Assert.isTrue(!processedSchedule, errorMessage);

processedSchedule = true;

tasks.add(this.registrar.scheduleFixedDelayTask(new FixedDelayTask(runnable, fixedDelay, initialDelay)));

}

String fixedDelayString = scheduled.fixedDelayString();

if (StringUtils.hasText(fixedDelayString)) {

if (this.embeddedValueResolver != null) {

fixedDelayString = this.embeddedValueResolver.resolveStringValue(fixedDelayString);

}

if (StringUtils.hasLength(fixedDelayString)) {

Assert.isTrue(!processedSchedule, errorMessage);

processedSchedule = true;

try {

fixedDelay = parseDelayAsLong(fixedDelayString);

}

catch (RuntimeException ex) {

throw new IllegalArgumentException(

"Invalid fixedDelayString value \"" + fixedDelayString + "\" - cannot parse into long");

}

tasks.add(this.registrar.scheduleFixedDelayTask(new FixedDelayTask(runnable, fixedDelay, initialDelay)));

}

}

// Check fixed rate

long fixedRate = scheduled.fixedRate();

if (fixedRate >= 0) {

Assert.isTrue(!processedSchedule, errorMessage);

processedSchedule = true;

tasks.add(this.registrar.scheduleFixedRateTask(new FixedRateTask(runnable, fixedRate, initialDelay)));

}

String fixedRateString = scheduled.fixedRateString();

if (StringUtils.hasText(fixedRateString)) {

if (this.embeddedValueResolver != null) {

fixedRateString = this.embeddedValueResolver.resolveStringValue(fixedRateString);

}

if (StringUtils.hasLength(fixedRateString)) {

Assert.isTrue(!processedSchedule, errorMessage);

processedSchedule = true;

try {

fixedRate = parseDelayAsLong(fixedRateString);

}

catch (RuntimeException ex) {

throw new IllegalArgumentException(

"Invalid fixedRateString value \"" + fixedRateString + "\" - cannot parse into long");

}

tasks.add(this.registrar.scheduleFixedRateTask(new FixedRateTask(runnable, fixedRate, initialDelay)));

}

}

// Check whether we had any attribute set

Assert.isTrue(processedSchedule, errorMessage);

// Finally register the scheduled tasks

synchronized (this.scheduledTasks) {

Set<ScheduledTask> regTasks = this.scheduledTasks.computeIfAbsent(bean, key -> new LinkedHashSet<>(4));

regTasks.addAll(tasks);

}

}

catch (IllegalArgumentException ex) {

throw new IllegalStateException(

"Encountered invalid @Scheduled method '" + method.getName() + "': " + ex.getMessage());

}

}

### **3、扫描SchedulingConfigurer实现类**

1、ScheduledAnnotationBeanPostProcessor实现了ApplicationListener接口，当工程启动好后调用onApplicationEvent方法，执行finishRegistration方法。

2、扫描所有的SchedulingConfigurer实现类，调用configureTasks回调函数添加定时任务。

3、调用registrar 的afterPropertiesSet方法。

@Override

public void onApplicationEvent(ContextRefreshedEvent event) {

if (event.getApplicationContext() == this.applicationContext) {

// Running in an ApplicationContext -> register tasks this late...

// giving other ContextRefreshedEvent listeners a chance to perform

// their work at the same time (e.g. Spring Batch's job registration).

finishRegistration();

}

}

private void finishRegistration() {

if (this.scheduler != null) {

this.registrar.setScheduler(this.scheduler);

}

if (this.beanFactory instanceof ListableBeanFactory) {

Map<String, SchedulingConfigurer> beans =

((ListableBeanFactory) this.beanFactory).getBeansOfType(SchedulingConfigurer.class);

List<SchedulingConfigurer> configurers = new ArrayList<>(beans.values());

AnnotationAwareOrderComparator.sort(configurers);

for (SchedulingConfigurer configurer : configurers) {

configurer.configureTasks(this.registrar);

}

}

if (this.registrar.hasTasks() && this.registrar.getScheduler() == null) {

Assert.state(this.beanFactory != null, "BeanFactory must be set to find scheduler by type");

try {

// Search for TaskScheduler bean...

this.registrar.setTaskScheduler(resolveSchedulerBean(this.beanFactory, TaskScheduler.class, false));

}

catch (NoUniqueBeanDefinitionException ex) {

if (logger.isTraceEnabled()) {

logger.trace("Could not find unique TaskScheduler bean - attempting to resolve by name: " +

ex.getMessage());

}

try {

this.registrar.setTaskScheduler(resolveSchedulerBean(this.beanFactory, TaskScheduler.class, true));

}

catch (NoSuchBeanDefinitionException ex2) {

if (logger.isInfoEnabled()) {

logger.info("More than one TaskScheduler bean exists within the context, and " +

"none is named 'taskScheduler'. Mark one of them as primary or name it 'taskScheduler' " +

"(possibly as an alias); or implement the SchedulingConfigurer interface and call " +

"ScheduledTaskRegistrar#setScheduler explicitly within the configureTasks() callback: " +

ex.getBeanNamesFound());

}

}

}

catch (NoSuchBeanDefinitionException ex) {

if (logger.isTraceEnabled()) {

logger.trace("Could not find default TaskScheduler bean - attempting to find ScheduledExecutorService: " +

ex.getMessage());

}

// Search for ScheduledExecutorService bean next...

try {

this.registrar.setScheduler(resolveSchedulerBean(this.beanFactory, ScheduledExecutorService.class, false));

}

catch (NoUniqueBeanDefinitionException ex2) {

if (logger.isTraceEnabled()) {

logger.trace("Could not find unique ScheduledExecutorService bean - attempting to resolve by name: " +

ex2.getMessage());

}

try {

this.registrar.setScheduler(resolveSchedulerBean(this.beanFactory, ScheduledExecutorService.class, true));

}

catch (NoSuchBeanDefinitionException ex3) {

if (logger.isInfoEnabled()) {

logger.info("More than one ScheduledExecutorService bean exists within the context, and " +

"none is named 'taskScheduler'. Mark one of them as primary or name it 'taskScheduler' " +

"(possibly as an alias); or implement the SchedulingConfigurer interface and call " +

"ScheduledTaskRegistrar#setScheduler explicitly within the configureTasks() callback: " +

ex2.getBeanNamesFound());

}

}

}

catch (NoSuchBeanDefinitionException ex2) {

if (logger.isTraceEnabled()) {

logger.trace("Could not find default ScheduledExecutorService bean - falling back to default: " +

ex2.getMessage());

}

// Giving up -> falling back to default scheduler within the registrar...

logger.info("No TaskScheduler/ScheduledExecutorService bean found for scheduled processing");

}

}

}

this.registrar.afterPropertiesSet();

}

### **4、添加定时任务到线程池**

1、afterPropertiesSet实际是在调用scheduleTasks方法安排任务

判断任务执行器是否存在，如果不存在则使用 2

2、Executors.newSingleThreadScheduledExecutor()

3、判断triggerTasks、cronTasks、fixedRateTasks、fixedDelayTasks是否存在，如果存在则addScheduledTask(scheduleTriggerTask(task))添加到taskScheduler，然后添加到scheduledTasks

@Override

public void afterPropertiesSet() {

scheduleTasks();

}

/\*\*

\* Schedule all registered tasks against the underlying

\* {@linkplain #setTaskScheduler(TaskScheduler) task scheduler}.

\*/

@SuppressWarnings("deprecation")

protected void scheduleTasks() {

if (this.taskScheduler == null) {

this.localExecutor = Executors.newSingleThreadScheduledExecutor();

this.taskScheduler = new ConcurrentTaskScheduler(this.localExecutor);

}

if (this.triggerTasks != null) {

for (TriggerTask task : this.triggerTasks) {

addScheduledTask(scheduleTriggerTask(task));

}

}

if (this.cronTasks != null) {

for (CronTask task : this.cronTasks) {

addScheduledTask(scheduleCronTask(task));

}

}

if (this.fixedRateTasks != null) {

for (IntervalTask task : this.fixedRateTasks) {

addScheduledTask(scheduleFixedRateTask(task));

}

}

if (this.fixedDelayTasks != null) {

for (IntervalTask task : this.fixedDelayTasks) {

addScheduledTask(scheduleFixedDelayTask(task));

}

}

}

用triggerTasks进行分析，其他任务类似：

1、首先从未解决的任务里面获取并移除当前任务

2、如果为空，则创建新的任务；从scheduleTasks()方法进来的时候已经存在任务

3、如果存在任务执行器，则调用方法taskScheduler.schedule安排任务，并返回一个future执行结果

@Nullable

public ScheduledTask scheduleTriggerTask(TriggerTask task) {

ScheduledTask scheduledTask = this.unresolvedTasks.remove(task);

boolean newTask = false;

if (scheduledTask == null) {

scheduledTask = new ScheduledTask(task);

newTask = true;

}

if (this.taskScheduler != null) {

scheduledTask.future = this.taskScheduler.schedule(task.getRunnable(), task.getTrigger());

}

else {

addTriggerTask(task);

this.unresolvedTasks.put(task, scheduledTask);

}

return (newTask ? scheduledTask : null);

}

private void addScheduledTask(@Nullable ScheduledTask task) {

if (task != null) {

this.scheduledTasks.add(task);

}

}